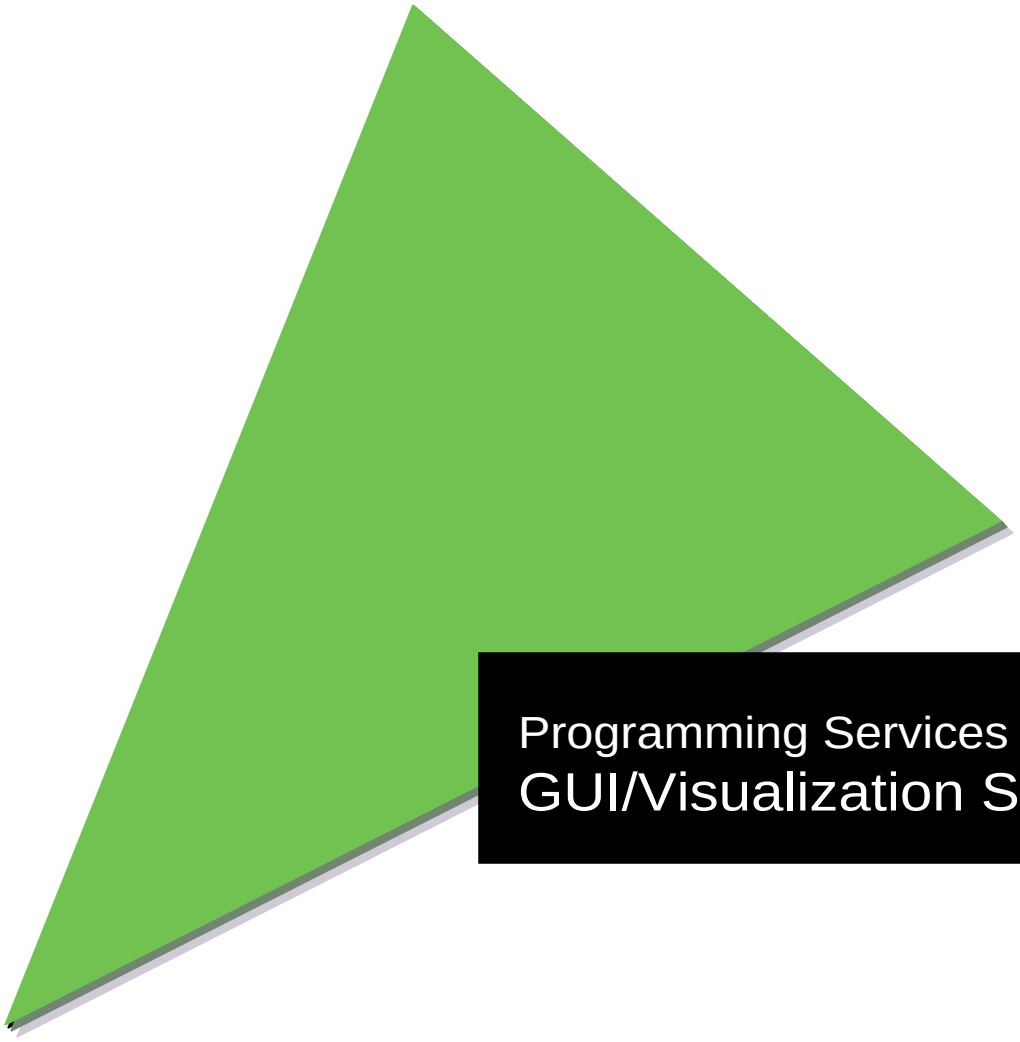


Programming Services Volume 3
GUI/Visualization Services

AccuSoft Corp.
www.accusoft.com

Chapter 1

Introduction

Chapter 1 - Introduction

A. Introduction To GUI & Visualization Services

There are eight libraries in VisiQuest 2001 that are based on X Windows. These libraries are available for use by *xvroutines*, and together provide the VisiQuest 2001 GUI and Visualization Services. The libraries in the GUI and Visualization Services rely on X Windows (X11R5 or X11R6), and the X Intrinsics Toolkit.

GUI and Visualization Services supports the creation and display of self-contained, self-maintaining *visual* and *GUI objects*. An object in this context is an abstract identifier; it may be actualized as a widget provided by one of the supported widget sets, or as a widget or gadget written specifically for VisiQuest 2001.

A *widget* is defined by Nye and O'Reilly [1] as "a pre-built user interface component." Such a GUI component is understood to have its own dedicated window. It is self-contained and self-maintaining. It automatically takes care of refreshing itself, resizing itself, and other "personal appearance" tasks; it knows how to deal with user input and, if applicable, has a predefined method of output.

The widgets used by GUI and Visualization Services have been written specifically for VisiQuest 2001. Some VisiQuest 2001 widgets have been written to support very specialized capabilities needed in visual objects; among many others, these include the image, zoom, pseudocolor, and animation widgets.

A *gadget* is defined by Nye and O'Reilly [2] as a "simplified widget which does not create a window." A number of the visual objects offered by GUI and Visualization Services are gadgets which were written specifically for VisiQuest 2001. Examples of such gadgets include plots, axes, lines, and circles. In each case, the specific gadget is generalized as a visual object.

The differentiation between a visual object and a GUI object is determined more by the use of the object than by any particular characteristic of the object itself. Visual objects are used for the display of graphics, images, annotations, plots, colormaps, and other types of data. In contrast, GUI objects provide mechanisms on a GUI that perform a specific I/O task, such as buttons, scrollbars, and lists.

Since GUI and Visualization Services must provide a consistent, abstract interface to objects that may be instantiated as either widgets or gadgets, it is necessary to use a common data type to implement the abstract application programming interface. The *xvobject* is the data type that represents [3] both GUI or visual objects. It hides the fact that a particular object might be instantiated as either a widget or a gadget. All functions that handle GUI and visual objects take the *xvobject* data type as the internal representation of the object. The calling application should not change, manipulate, or even see the contents of the underlying data structure.

---

[1] *X ToolKit Intrinsics Programming Manual*, by Adrian Nye and Tim O'Reilly

[2] *X ToolKit Intrinsics Programming Manual*, by Adrian Nye and Tim O'Reilly

[3] This technique is used by several different libraries in VisiQuest 2001 for the same reason; for example, the *kobject* used by Data Services and the (differing) *kobject* used by the data transport mechanism. By convention, abstract data types that are to be hidden from the calling application are called "kobject" if they are *not* related to visual display; "xvobject" if they are related to visual object.

These are the eight libraries that make up GUI and Visualization Services:
• The *xvwidgets* library (*libxvw.a, libxvw.so*) of the Design toolbox
• The *xvobjects* library (*libxvobj.a, libxvobj.so*) of the Design toolbox
• The *xvimage* library (*libxvi.a, libxvi.so*) of the Envision toolbox
• The *xvplot* library (*libxvp.a, libxvp.so*) of the Envision toolbox
• The *xvannotations* library (*libxva.a, libxva.so*) of the Envision toolbox
• The *xvlang* library (*libxvl.a, libxvl.so*) of the Imagine toolbox
• The *xvforms* library (*libxvf.a, libxvf.so*) of the Design toolbox
• The *xvutils* library (*libxvu.a, libxvu.so*) of the Design toolbox

Each library has its own chapter in this volume of the Program Services Manual.

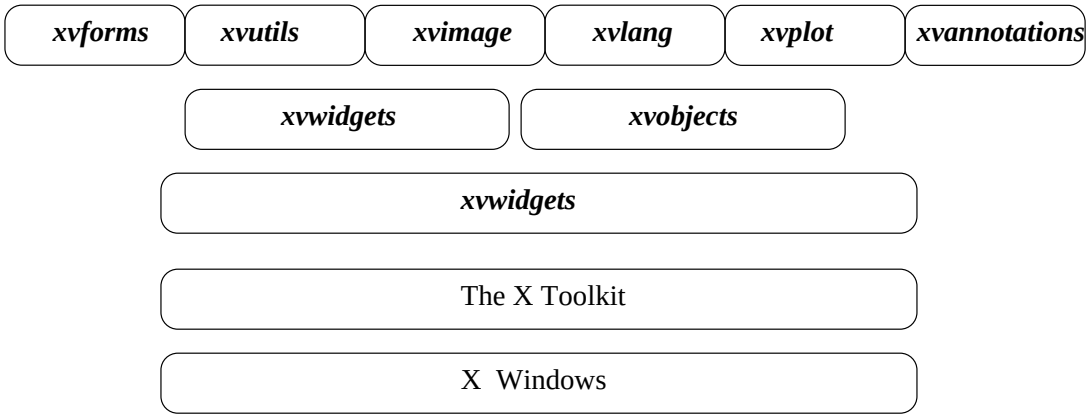


Figure 1: Diagram depicting the relationship between libraries in VisiQuest Programming Services, X libraries, and the different supported widget sets.

A.1. The xvwidgets Library

The *xvwidgets* library is the lowest level library; the other five libraries all depend on routines available in *xvwidgets*. This library contains a number of routines that are front ends to the X Toolkit, which should be called by the application instead of the X Toolkit calls, in order to ensure that an application will support all three widget sets. Callbacks, event handlers, action handlers, timeouts, and input handlers are all installed on GUI and visual objects using the *xvwidgets* library. Characteristics of GUI and visual objects are controlled by setting and retrieving attributes using functions provided by the *xvwidgets* library. Consistent use of the *xvwidgets* library ensures that xvroutines will have a standard of functionality and a familiar method of operation regardless of whether a particular object being used is instantiated as a widget or a gadget.

The *xvwidgets* library also contains basic GUI objects that are the VisiQuest 2001 versions of the most common widgets, such as the button object, the label object, the text dialog object, the scrollbar object, the list object, the menu object, the menubutton object, and the viewport object.

A key element of the *xvwidgets* library is the VisiQuest 2001 Manager object, which supports the direct manipulation of GUI and visual objects. Moreover, the manager object allows objects to display *internal menuforms*. These menuforms allow the user to set attributes of the objects without additional support from the calling program.

widgets in the xvwidgets library

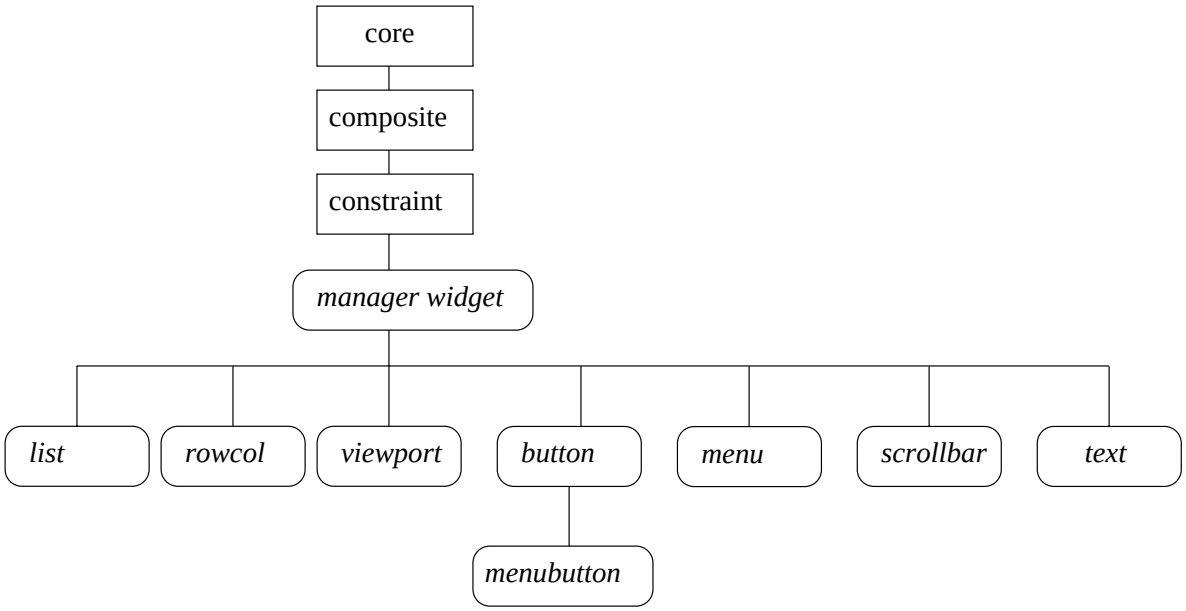


Figure 2: This is the inheritance tree of the GUI objects in the *xvwidgets* library that are implemented as *widgets*. The Core, Composite, and Constraint widgets are provided by the X Toolkit; the rest of the widgets are provided by the *xvwidgets* library.

gadgets in the xvwidgets library

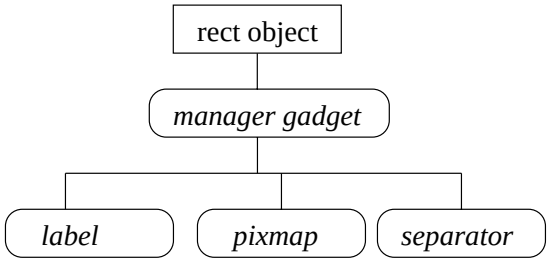


Figure 3: The inheritance tree of the GUI objects in the *xvwidgets* library that are implemented as *gadgets*. The Rect object is provided by the X Toolkit; the rest of the gadgets are provided by the *xvwidgets* library.

A.2. The xvobjects Library

The *xvobjects* library also contains GUI objects that were written specifically for VisiQuest 2001. While the *xvwidgets* library supports the most general and commonly-used objects, the *xvobjects* library contains GUI objects that are more specific in nature. They have a variety of purposes, but all deal with I/O rather than image or graphics display. Some of these support application reporting, specifically the info object, warn

object, and error object. The help object is used for display of online help pages. Other objects are used to obtain user input of a particular type, such as the inputfile, outputfile, float, integer, and double objects. Other objects provided meet various needs of VisiQuest 2001 application graphical user interfaces, such as the connection object which is used by VisiQuest to connect glyphs, the console object which is used by both VisiQuest and Craftsman to echo output of spawned processes, or the notifywindow object which is used by many VisiQuest 2001 applications to indicate when the program is "working."

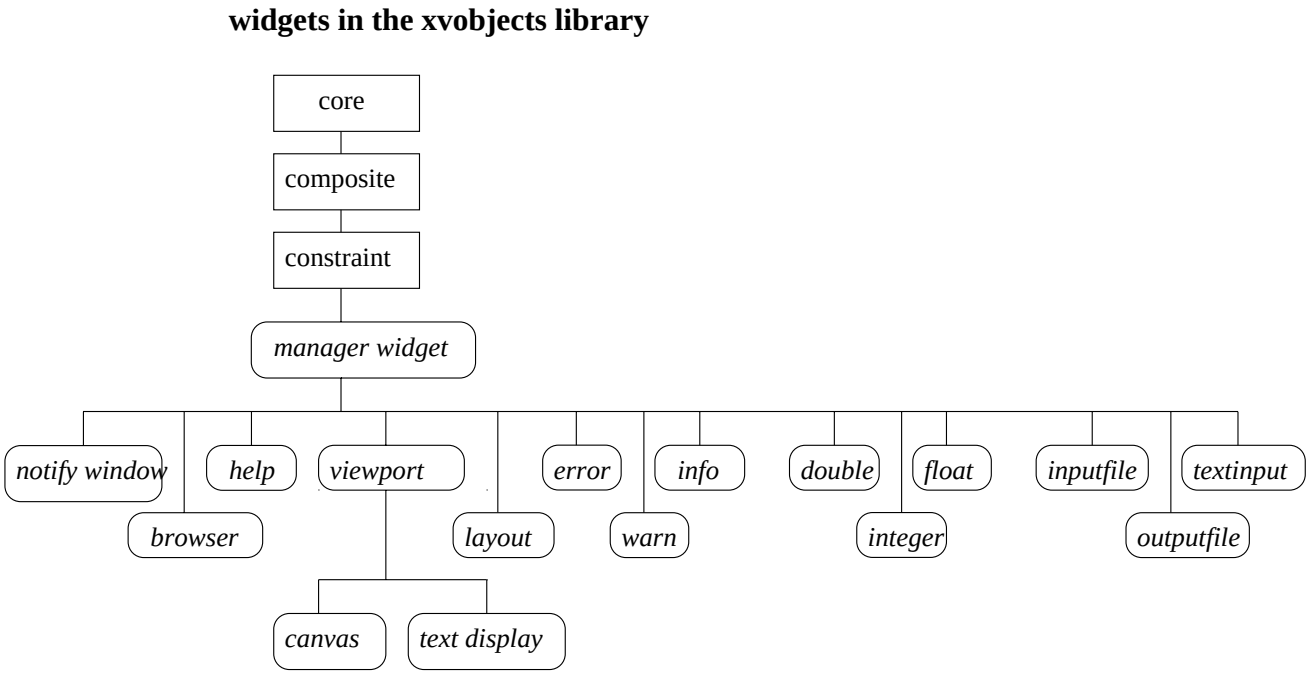


Figure 4: This is the inheritance tree of the GUI objects in the *xvobjects* library that are implemented as *widgets*. The Core, Composite, and Constraint widgets are provided by the X Toolkit; the manager widget and the viewport widget are provided by the *xvwidgets* library; the rest of the widgets are provided by the *xvobjects* library.

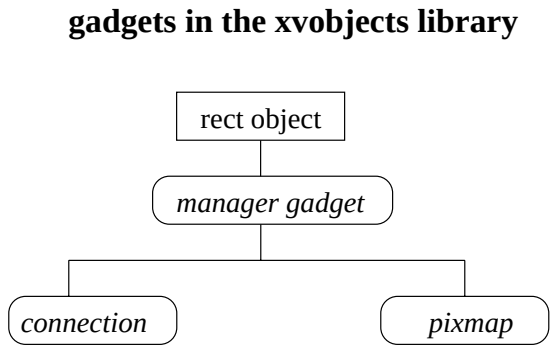


Figure 5: The inheritance tree of those GUI objects in the *xvobjects* library that are implemented as *gadgets*. The Rect object is provided by the X Toolkit; the manager gadget is part of the *xvwidgets* library; the other gadgets are provided by the *xvobjects* library.

A.3. The xvimage Library

The *xvimage* library provides utilities with which you may create and display images in a number of ways. These include the pan icon, the zoom widget, and the animation object, among others. There are also a number of objects with which you may examine and modify the colormap of an object. The threshold object, the pseudo object, the palette, and the colorcell are among them. The objects in the *xvimage* library are all widgets.

widgets in the xvimage library

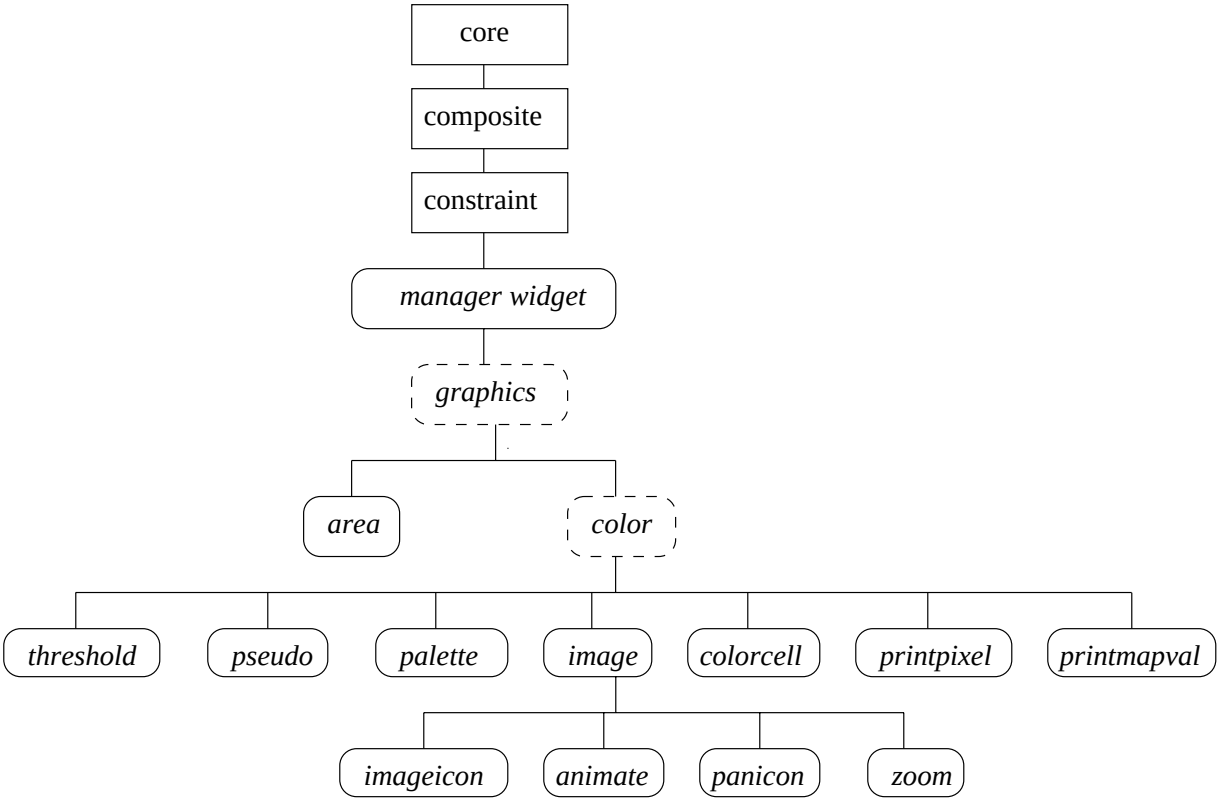


Figure 6: The *xvimage* library offers a variety of visual objects for image display. This diagram shows the inheritance tree of those visual objects in the *xvimage* library; they are all implemented as *widgets*. The Core, Composite, and Constraint widgets are provided by the X Toolkit. The rest of the widgets are provided by the *xvisual* library, except for the manager and the rowcol widget, both of which are constraint widgets found in the *xvwidgets* library. Note that the color class (depicted with dashed lines) serves only to allow subclassing by other objects; it cannot be created as an object directly.

A.4. The xvannotations Library

The *xvannotations* library provides utilities with which you may create and display visual objects which would typically be classified as annotations. These include such primitives as lines, circles, and rectangles. The objects in the *xvannotations* library are all gadgets.

gadgets in the xvisual library

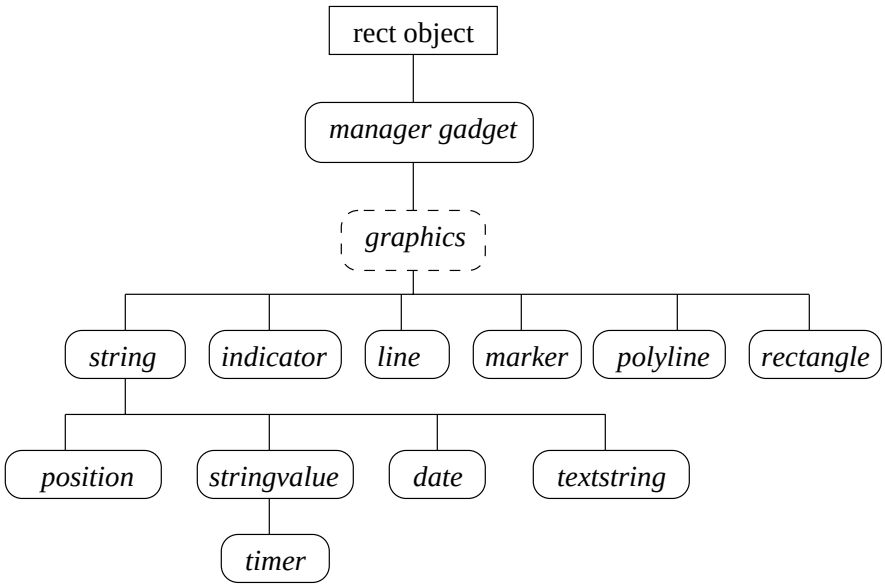


Figure 7: The inheritance tree of those visual objects in the *xvannotations* library, which are all implemented as *gadgets*. The Rect object is provided by the X Toolkit; the rest of the gadgets (with the exception of the manager gadget, provided by the *xvwidgets* library) are provided by the *xvannotations* library. Note that the graphics class and the color class (depicted with dashed lines) serve only to allow subclassing by other objects; they cannot be created as objects directly.

A.5. The xvplot Library

The *xvplot* library provides utilities with which you may create and display plots and axes. The gadgets provided include the 2D axis object and the 2D plot object.

gadgets in the xvplot library

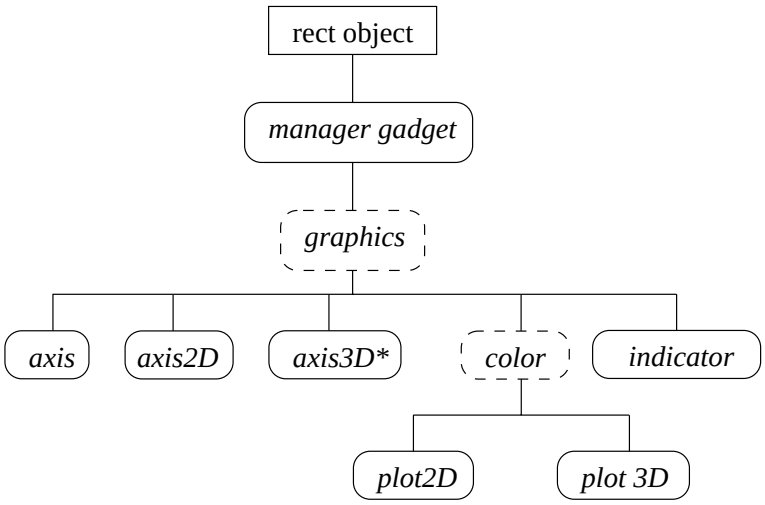


Figure 8: The inheritance tree of the gadgets in the *plot* library that are implemented as *gadgets*. The Rect object is provided by the X Toolkit; the rest of the gadgets are provided by the *xvplot* library. Note that the graphics class (depicted with dashed lines) serves only to allow subclassing by other objects; it cannot be created as an object directly.

A.6. The xvlang Library

The *xvlang* library of VisiQuest 2001 supports an object- oriented approach to the design and implementation of visual programming languages. The visual programming objects offered by the *xvlang* visual programming toolkit are used in the VisiQuest 2001 visual programming language, VisiQuest.

The visual programming language toolkit offered by the *xvlang* library follows an object-oriented approach to the design and development of visual programming languages. Use of the visual programming objects available in the toolkit allows flexibility and reusability to experiment with different visual programming paradigms, and offers the possibility of adapting existing models to meet the needs of new visual languages.

By providing a visual programming object which addresses each key component of the visual programming environment, *xvlang* decouples the complexity of those visual programming components from the visual programming environment itself. This allows the developer of the visual programming language to concentrate on the visual programming paradigm to be developed, rather than on the functionality of the various components necessary to the language.

widgets in the xvlang library

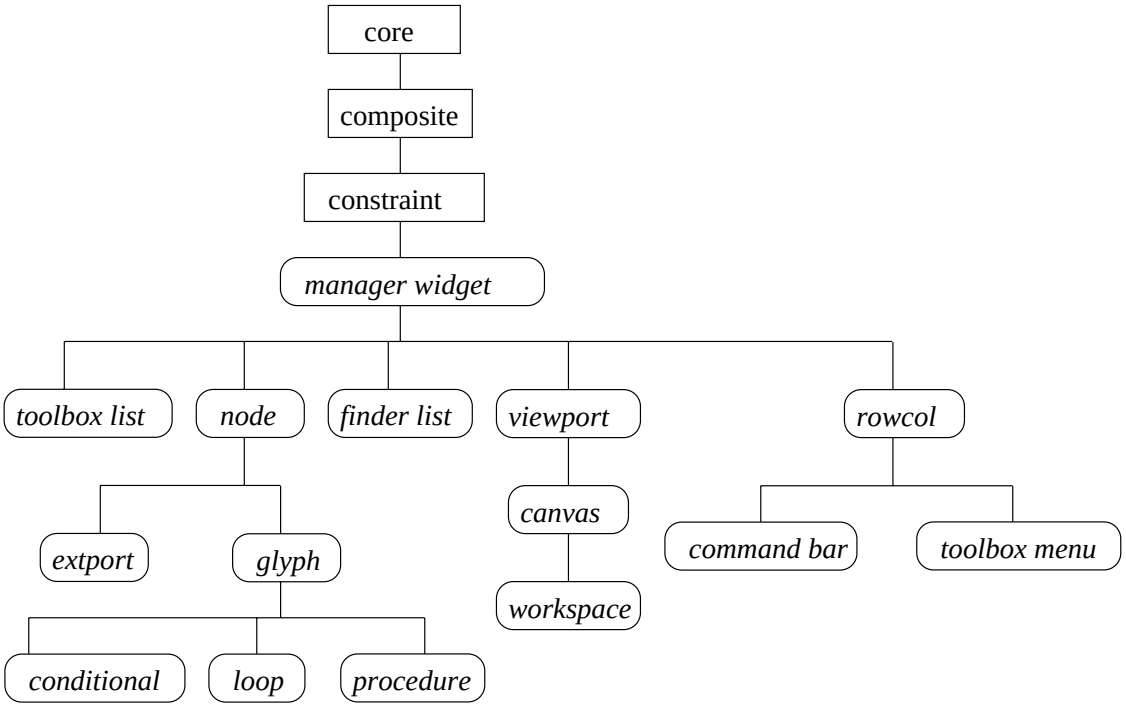


Figure 9: The *xvlang* library provides visual objects specifically for use by visual languages. They are used by cantata, and are made publically available to developers. This diagram shows the inheritance tree of those visual objects in the *xvlang* library which are all implemented as *widgets*. The Core, Composite, and Constraint widgets are provided by the X Toolkit. The manager widget, rowcol object, and viewport object are part of the *xvwidgets* library, while the canvas object is found in the *xvobjects* library. The rest of the widgets depicted are provided by the *xvlang* library.

A.7. The xvforms Library

The *xvforms* library provides all the functionality necessary to create and maintain the Graphical User Interface (GUI) of an application, where that GUI is defined by a User Interface Specification (UIS) file. The main GUI drivers with the minimum number of correct calls to the *xvforms* library are automatically generated for an *xvroutine*; thus, it is not necessary to construct the GUI driver of an *xvroutine* from scratch. However, calls to some routines available in the *xvforms* may be added frequently to an application, such as calls to *xvf_set_attribute()*, which allows the *xvroutine* to modify its GUI during runtime.

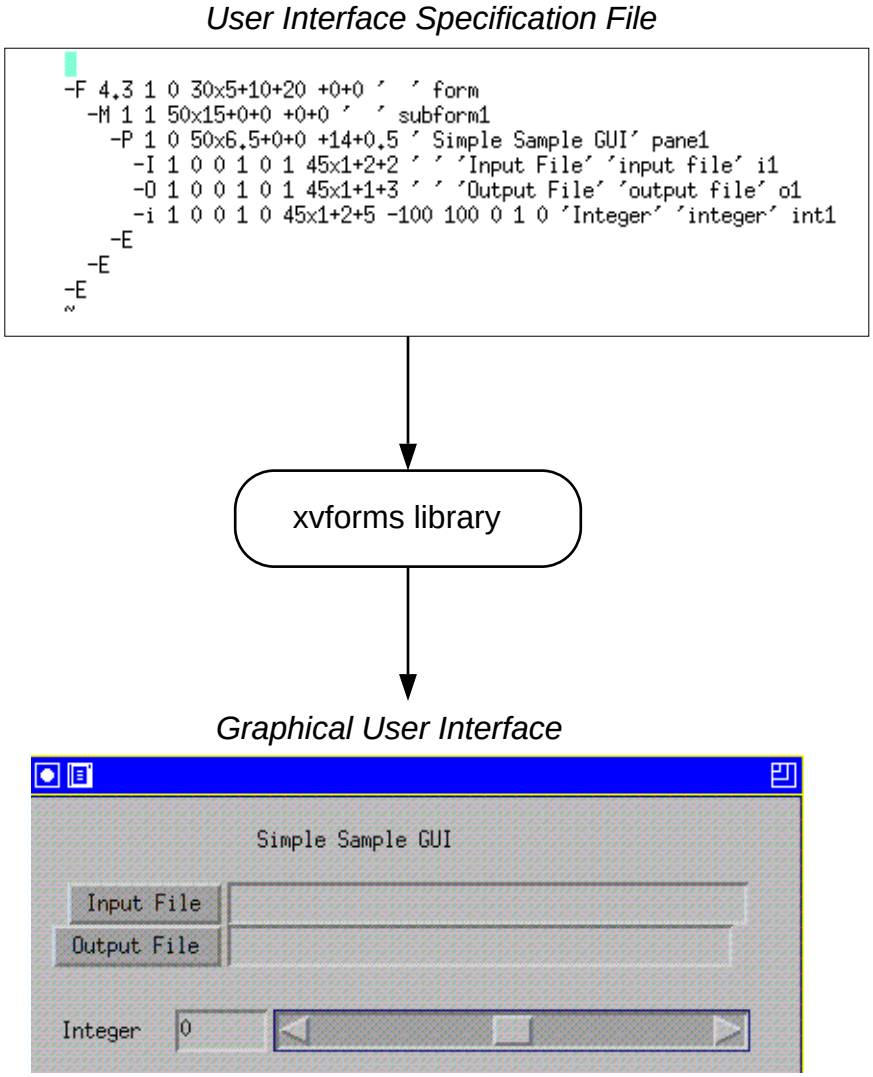


Figure 10: The main purpose of the *xvforms* library is to take a UIS file and create the GUI that it defines. It runs and maintains the GUI; the application may dynamically change its GUI with a call to *xvf_set_attribute()*.

A.8. The xvutils Library

The *xvutils* library contains a number of utility functions which are generally used to augment the GUI created with the *xvforms* library. These utilities are all which are compound combinations of GUI and visual objects that either prompt the user for information, or display information until they are acknowledged. Such utilities are commonly used by *xvroutines*, and include such items as file browsers, list objects, pop-up error messages, and file display objects.

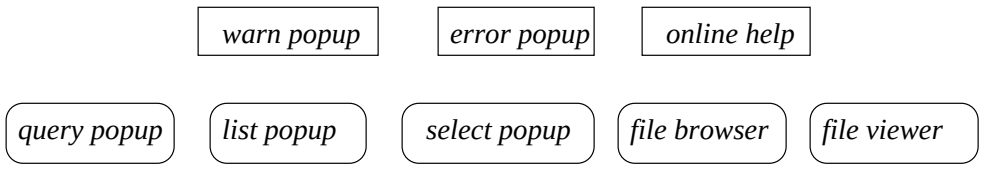


Figure 11: The *xvutils* library provides a number of utility functions that produce popup dialogs for use with VisiQuest xvroutines. Some prompt for information (those on the bottom row), while others simply require acknowledgement (those on the top row). All appear in popup windows that are independent from the GUI of the application; most block for input.

Table of Contents

This page left intentionally blank

Chapter 2

Xvwidgets

Chapter 2 - Xvwidgets

A. Introduction

The *xvwidgets* (*libxvw.a*) library is the lowest-level library of the VisiQuest 2001 GUI & Visualization Services; it has several functions.

The *xvwidgets* library supports the management of GUI and visual objects. Some GUI and visual objects are instantiated as *widgets*, meaning that they have their own dedicated window. Other GUI and visual objects are instantiated as *gadgets*, or widgets that do not have their own window, but are displayed on their parent's window. The X Toolkit provides solid support for widgets; support for gadgets, however, is not as well developed. To the VisiQuest Toolbar, however, the differences between widgets and gadgets are minimized or eliminated altogether with the application programming interface provided by the *xvwidgets* library. The same *xvwidgets* routines are used to add callbacks, event handlers, action procedures, input handlers, and timeouts on all GUI and visual objects, regardless of whether they are implemented as widgets or gadgets.

Xvwidgets contains the most basic of the GUI objects, which are most frequently seen on VisiQuest 2001 Graphical User Interface (GUI). These include the VisiQuest 2001 Manager object as well as the button, label, list, me nu, menubutton, pixmap, rowcol, scrollbar, text, and viewport objects.

The most specialized widget supported by GUI & Visualization Services is the VisiQuest 2001 *Manager* widget. Use of the Manager widget by the applicatio n program allows the user to have direct control over the internally-defined attributes of the GUI and visual objects, without intervention on the part of the application. In addition, the manager object allows direct manipulation of both GUI and visual objects which are its children. This makes it possible for an application to incorporate a significant amount of graphical and image display, *and* to provide complete support for the interactive modification of all the attributes associated with the objects displayed, with only a few lines of code.

Functions for GUI object creation

- *xvw_create_button()* - create a button object
- *xvw_create_label()* - create a label object
- *xvw_create_list()* - create a list object
- *xvw_create_manager()* - create a VisiQuest Manager object
- *xvw_create_menu()* - create a menu object
- *xvw_create_menubutton()* - create a menubutton object
- *xvw_create_pixmap()* - create a pixmap object
- *xvw_create_rowcol()* - create a row-col object
- *xvw_create_scrollbar()* - create a scrollbar object
- *xvw_create_text()* - create a text object
- *xvw_create_viewport()* - create a viewport object

As implied above, another critical function of the *xvwidgets* library is to mediate between the application and the X Toolkit when necessary. There are a number of utility routines in the *xvwidgets* library that are front ends for X Toolkit functions. Whenever available, these functions *must* be called instead of their X Toolkit

counterparts, in order to ensure that the application program work properly.

GUI & Visualization Services Function	X Toolkit Function
xvw_add_action()	XtOverrideTranslations(), XtAugmentTranslations()
xvw_add_callback()	XtAddCallback(), XtAddCallbacks()
xvw_add_detectfid()	XtAppAddInput()
xvw_add_event()	XtAppAddEventHandler()
xvw_add_timeout()	XtAppAddTimeOut()
xvw_appcontext()	XtAppContext()
xvw_call_callback()	XtCallCallbacks()
xvw_check_managed()	XtIsManaged()
xvw_check_realized()	XtIsRealized()
xvw_check_sensitive()	XtIsSensitive()
xvw_check_subclass()	XtIsSubclass()
xvw_class()	XtClass()
xvw_colormap()	DefaultColormap()
xvw_create(), xvw_create_xxx()	XtCreateWidget(), XtCreateManagedWidget()
xvw_create_application_shell(), xvw_create_transient_shell()	XtAppCreateShell()
xvw_display()	XtDisplay()
xvw_destroy()	XtDestroyWidget()
xvw_get_attribute(s)()	XtGetValues(), XtVaGetValues()
xvw_initialize()	XtToolkitInitialize()
xvw_manage()	XtManageChild()
xvw_map()	XtMapWidget()
xvw_name()	XtName()
xvw_parent()	XtParent()
xvw_realize()	XtRealizeWidget()
xvw_remove_action()	XtRemoveActions()
xvw_remove_callback()	XtRemoveCallback()
xvw_remove_event()	XtRemoveEventHandler()
xvw_remove_timeout()	XtRemoveTimeout()
xvw_rootwindow()	RootWindow()
xvw_sensitive()	XtSetSensitive()
xvw_set_attribute(s)()	XtSetValues(), XtVaSetValues()
xvw_translate_coords()	XtTranslateCoords()
xvw_unmanage()	XtUnmanageChild()
xvw_unmap()	XtUnmapWidget()
xvw_unrealize()	XtUnrealizeWidget()
xvw_visual()	DefaultVisual()
xvw_window()	XtWindow()

In addition, there are a number of utility functions that are provided simply for the convenience of the developer; you are encouraged to become familiar with these functions and use them whenever appropriate.

Available Functions

In alphabetical order, the functions available in the *xvwidgets* library include:

• *xvw_activate_menu()* - pop up the internal menuform for an object.
• *xvw_activated_menu()* - see if the internal menuform for an object is currently mapped
• *xvw_add_action()* - add an action handler to an object
• *xvw_add_callback()* - add a callback to a GUI object
• *xvw_add_detectfid()* - add (fid) input handler to an object
• *xvw_add_detectfile()* - add a (file) detect handler to an object
• *xvw_add_event()* - add an event handler to an object
• *xvw_add_timeout()* - add a timeout to an object
• *xvw_appcontext()* - return the application context associated with a object
• *xvw_busy()* - set an object to be busy or not busy
• *xvw_check_managed()* - see if an object is managed
• *xvw_check_mapped()* - see if an object is mapped
• *xvw_check_menuactive()* - see if an object's internal menuform is displayed
• *xvw_check_menuexist()* - check if an object has an internal menuform
• *xvw_check_realized()* - see if an object is realized
• *xvw_check_sensitive()* - see if an object is sensitive
• *xvw_check_subclass()* - check the subclass of an object
• *xvw_check_toplevel()* - see if object specified is a toplevel, or see if a toplevel exists
• *xvw_check_visible()* - see if an object is visible
• *xvw_children()* - get the children of an object
• *xvw_class()* - get the class of the object
• *xvw_colormap()* - get the colormap associated with a object
• *xvw_create()* - create a new object
• *xvw_create_application_shell()* - create an application shell object
• *xvw_create_transient_shell()* - create a transient shell object
• *xvw_destroy()* - destroy an object
• *xvw_display()* - returns the display associated with a object
• *xvw_duplicate()* - duplicate an object
• *xvw_font()* - return the font being used by a object
• *xvw_fontname()* - return the font name being used by an object
• *xvw_geometry()* - get the geometry of an object
• *xvw_get_attribute()* - get a single attribute of an object
• *xvw_get_attributes()* - get attributes from an object (variable argument list)
• *xvw_inactivate_menu()* - pop down the internal menuform for an object.
• *xvw_initialize()* - initialize the xvwidgets library
• *xvw_insert_event()* - insert an event handler into an object's event list.
• *xvw_lower()* - lower an object
• *xvw_manage()* - manage an object
• *xvw_map()* - map an object
• *xvw_name()* - get the name of the object
• *xvw_numchildren()* - get the number of children of an object
• *xvw_object()* - get the object associated with a particular widget
• *xvw_parent()* - get the parent of an object
• *xvw_place()* - place an object on the screen
• *xvw_raise()* - raise an object
• *xvw_realize()* - realize an object

• *xvw_refresh()* - refreshes an object
• *xvw_remove_action()* - remove an action handler from an object
• *xvw_remove_callback()* - remove a callback from a GUI object
• *xvw_remove_detectfid()* - remove (fid) input handler from an object
• *xvw_remove_detectfile()* - remove a (file) detect handler from an object
• *xvw_remove_event()* - remove an event handler from an object
• *xvw_remove_timeout()* - removes a timeout from an object
• *xvw_rootwindow()* - get the root window associated with an object
• *xvw_screen()* - return the screen associated with a object
• *xvw_screennum()* - return the screen number associated with an object
• *xvw_sensitive()* - sensitize or de-sensitize an object
• *xvw_set_attribute()* - set a single attribute on an object
• *xvw_set_attributes()* - set attributes on an object (variable argument list)
• *xvw_sort()* - sort a list of objects
• *xvw_toplevel()* - get the toplevel object of an object
• *xvw_unmanage()* - unmanage an object
• *xvw_unmap()* - unmap an object
• *xvw_unrealize()* - un-realize an object
• *xvw_visual()* - get the visual associated with an object
• *xvw_vset_attributes()* - set attributes on an object (variable argument list)
• *xvw_widget()* - get the widget (or gadget) associated with an object
• *xvw_window()* - get the window associated with an object

A.1. xvw_initialize() — *initialize the xvwidgets library*

Synopsis

```
Display *xvw_initialize(
    void (*menu_handler)(void))
```

Input Arguments

`menu_handler`

specifies the menu handling routines. May be one of:

XVW_MENUS_XVFORMS - Enable use of xvforms as internal menus
XVW_MENUS_NONE - Disable internal menus

Returns

The newly opened display on success, NULL on failure

Description

Opens the display, sets the widget set of choice, and initializes the xvwidgets library prior to use. A call to this routine *must* be made before any other calls to routines in GUI & Visualization services are made.

B. General Attributes Of GUI & Visual Objects

The characteristics of GUI and visual objects that can be specified by the application are called object *attributes*. Object *resources* are pairs of names and values that indicate the current setting of object attributes. Attributes of GUI and visual objects can be set and retrieved one at a time using *xvw_set_attribute()* and *xvw_get_attribute()*, or can be set and retrieved in groups using the variable-argument *xvw_set_attributes()* and *xvw_get_attributes()* (see the section entitled, "Setting And Getting Attributes").

The *xvwidgets* library allows certain attributes to be set and retrieved on all visual and GUI objects. When using general attributes, you have to use your common sense to determine which attributes are applicable to which objects. Setting or getting a particular attribute is only meaningful if the attribute applies to the object in question. For example, it should be expected that setting `XVW_FONT_NAME` on a scrollbar object is meaningless. However, these attributes commonly apply to all or most GUI and visual objects.

B.1. Pixel Geometry

The following attributes are used to specify general object geometry in *pixels*. All take integer values, and will take precedence over any counterparts specified in character widths (see "Character Geometry").

Summary of Pixel Geometry Attributes

Attribute	Description
`XVW_BORDER_WIDTH`	Specifies the size of the border, in pixels. A value of zero (0) indicates that no border is desired.
`XVW_HEIGHT`	The height of the object in pixels. Will take precedence over `XVW_CHAR_HEIGHT`.
`XVW_WIDTH`	The width of the object in pixels. Will take precedence over `XVW_CHAR_WIDTH`.
`XVW_XPOSITION`	The X position with respect to the upper left hand corner of the parent, in pixels. Takes precedence over `XVW_CHAR_XPOS`.
`XVW_YPOSITION`	The Y position with respect to the upper left hand corner of the parent, in pixels. Takes precedence over `XVW_CHAR_YPOS`.

Descriptions of Pixel Geometry Attributes

Attribute (Resource Name)	Type	Default	Legal Values
`XVW_BORDER_WIDTH` (`borderWidth`)	`int`	`1`	values >= 0
`XVW_HEIGHT` (`height`)	`int`	`calculated`	values > 0
`XVW_WIDTH` (`width`)	`int`	`calculated`	values > 0

Descriptions of Pixel Geometry Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_XPOSITION (xposition)	int	0	values >= 0
XVW_YPOSITION (yposition)	int	0	values >= 0

B.2. Character Geometry

In many cases involving a visual object that contains text, it is more convenient to specify the geometry of a object in terms of *character size* rather than pixels. The actual sizes involved will vary according to the font being used by the application. The following attributes take float values. They will be over-ridden by their counterparts specified in numbers of pixels (see "Pixel Geometry").

Summary of Character Geometry Attributes	
Attribute	**Description**
XVW_CHAR_HEIGHT	The height of the object in characters.
XVW_CHAR_MAX_HEIGHT	The maximum height of the object in characters.
XVW_CHAR_MAX_WIDTH	The maximum width of the object in characters.
XVW_CHAR_MIN_HEIGHT	The minimum height of the object in characters.
XVW_CHAR_MIN_WIDTH	The minimum width of the object in characters.
XVW_CHAR_WIDTH	The width of the object in characters.
XVW_CHAR_XPOS	The X position in characters, with respect to the upper left hand corner of the parent.
XVW_CHAR_YPOS	The Y position in characters, with respect to the upper left hand corner of the parent.

Descriptions of Character Geometry Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_CHAR_HEIGHT (charHeight)	float	calculated	values > 0.0
XVW_CHAR_MAX_HEIGHT (charMaxHeight)	float	calculated	values > 0.0
XVW_CHAR_MAX_WIDTH (charMaxWidth)	float	calculated	values > 0.0
XVW_CHAR_MIN_HEIGHT (charMinHeight)	float	calculated	values > 0.0
XVW_CHAR_MIN_WIDTH (charMinWidth)	float	calculated	values > 0.0

Descriptions of Character Geometry Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_CHAR_WIDTH (charWidth)	float	calculated	values > 0.0
XVW_CHAR_XPOS (charXpos)	float	0.0	values >= 0.0
XVW_CHAR_YPOS (charYpos)	float	0.0	values >= 0.0

B.3. Colors, Fonts, and Cursors

You may set the foreground, background, and border colors using either the name string of a color or a pixel value. Names of valid color name strings can be found by looking at /usr/lib/X11/rgb.txt, but be aware that the presence of a color name string in this file does not guarantee that the color will be available on a particular X server. You can specify the font for visual objects that involve text; cursors can be defined for any visual object.

Summary of General Attributes	
Attribute	**Description**
XVW_BACKGROUND	The pixel value that defines the desired color for the background.
XVW_BACKGROUND_COLOR	Provide the string that specifies the name of the desired background color. Will be over-ridden by XVW_BACKGROUND_PIXEL if both are set.
XVW_BACKGROUND_PIXEL	See XVW_BACKGROUND it is the same thing.
XVW_BACKGROUND_PIXMAP	This is an (optional) pixmap that appears as the background of an object, instead of the background color. Candidates for the value of this attribute may be created with the use of *XCreatePixmap()*; see *The Xlib Reference Manual* by O'Reilly and Associates. Note that this attribute is mutually exclusive with XVW_BACKGROUND_PIXMAPFILE; specify one or the other, not both.
XVW_BACKGROUND_PIXMAPFILE	This is the file defining an (optional) pixmap that appears as the background of an object, instead of the background color.
XVW_BORDER	Provide the pixel value that specifies the desired border color.
XVW_BORDER_COLOR	The string that specifies the name of the desired border color. Will be overridden by XVW_BORDER_PIXEL if both are set.
XVW_BORDER_PIXEL	See XVW_BORDER; it is the same thing.

Summary of General Attributes	
Attribute	**Description**
XVW_CHAR_XSNAP	This attribute only applies to *constraint* objects, such as manager objects, canvas objects, rowcol objects, and so on, that support layout and direct manipulation of children. With interactive movement of selected child objects, the child objects can be made to "snap" to the new position, aiding the user in more precise interactive layout of objects. This attribute specifies, in characters, the horizontal increments of the implied grid (visible or not) to which children will "snap". For example, if XVW_CHAR_XSNAP is set to 0.5, then objects will "snap" to positions in half-character-width increments.
XVW_CHAR_YSNAP	This attribute only applies to *constraint* objects, such as manager objects, canvas objects, rowcol objects, and so on, that support layout and direct manipulation of children. With interactive movement of selected child objects, the child objects can be made to "snap" to the new position, aiding the user in more precise interactive layout of objects. This attribute specifies, in characters, the vertical increments of the implied grid (visible or not) to which children will "snap". For example, if XVW_CHAR_YSNAP is set to 1.0, then objects will "snap" to positions in one-character-height increments.
XVW_COLORMAP	This is the X11 *colormap* that is used to display the visual object. Colormaps are discussed in detail in in *The Xlib Programming Manual* by Adrian Nye. Note that the application should generally *not* attempt to set the colormap; it should instead allow the visual object to choose the most appropriate colormap.
XVW_CURSOR	This attribute specifies the cursor to be used when the pointer is inside the visual object. The Cursor structure that must be passed for this attribute can be obtained with *XCreateGlyphCursor(), XCreatePixmapCursor(), XDefineCursor()*, or *XCreateFontCursor()*. See *The Xlib Reference Manual*, by Adrian Nye for definitions of these XLib calls; Appendix I on "The Cursor Font" may also be helpful.

Summary of General Attributes	
Attribute	**Description**
XVW_CURSORNAME	A number of standard cursors are made available with X Windows. The names of these predefined cursors are found in <X11/cursor-font.h>. The XVW_CURSOR_NAME attribute allows you to specify a predefined cursor by its name. Accepted cursor names include: "X_cursor", "array", "based_arrow_down", "based_arrow_up", "boat", "bogosity", "bottom_left_corner", "bottom_right_corner", "bottom_side", "bottom_tee", "box_spiral", "center_ptr", "circle", "clock", "coffee_mug", "cross", "cross_reverse", "crosshair", "diamond_cross", "dot", "dotbox", "double_arrow", "draft_large", "draft_small", "draped_box", "exchange", "fleur", "gobbler", "gumby", "hand1", "hand2", "heart", "icon", "iron_cross", "left_ptr", "left_side", "left_tee", "left_button", "ll_angle", "lr_angle", "man", "middlebutton", "mouse", "pencil", "pirate", "plus", "question_arrow", "right_ptr", "right_side", "right_tee", "rightbutton", "rtl_logo", "sailboat", "sb_down_arrow", "sb_h_double_arrow", "sb_left_arrow", "sb_right_arrow", "sb_v_double_arrow", "shuttle", "sizing", "spider", "spraycan", "star", "target", "tcross", "top_left_arrow", "top_left_corner", "top_right_corner", "top_side", "top_tee", "trek", "ul_angle", "ul_umbrella", "ur_angle", "watch", "xterm", and "num_glyphs". See Appendix I of *The Xlib Reference Manual* by O'Reilly & Associates for illustrations depicting the available standard cursors.
XVW_DEPTH	This is the number of planes that are to be used to represent grey scales or color within a visual object. It determined by the visual being used (see XVW_VISUAL)

XVW_DESTROY	If necessary, *xvw_add_callback()* may be used to install a callback on any GUI or visual object which will be fired when the object is destroyed. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(object, XVW_DESTROY,`
	`destroy_cb, client_data);`

XVW_FONT	The font to be used with any text associated with the visual object. The XFontStruct that must be passed for this attribute can be obtained with *XLoadQueryFont()*; see Section 6.2 of *The Xlib Programming Manual* by Adrian Nye. Note that this attribute is mutually exclusive with XVW_FONTNAME; use one or the other, not both. Available fonts for a particular workstation can be obtained from the command line with `%xlsfonts`, or from an application program with *XListFonts()*.
XVW_FONTNAME	This attribute specifies the name of the font to be used with any text associated with the visual object. A list of the available fonts on a particular X server can be obtained with the command, "xlsfonts". Fonts are listed in the "fonts.dir" file in /usr/lib/X11/fonts/misc, /usr/lib/X11/fonts/75dpi, and /usr/lib/X11/fonts/100dpi. The various fonts used by X11 are defined in these directories.
XVW_FOREGROUND	The pixel value that defines the desired color for the foreground.

Summary of General Attributes	
Attribute	**Description**
XVW_FOREGROUND_COLOR	Provide the string that specifies the name of the desired foreground color. Will be over-ridden by XVW_FOREGROUND_PIXEL if both are set.
XVW_FOREGROUND_PIXEL	See XVW_FOREGROUND; it is the same thing.
XVW_MAPPED	Dictates whether or not the object is mapped. If the object has not been created, its initial state may be *mapped* (displayed to the screen immediately) or *unmapped* (not displayed until explicitly told to be displayed, with a call to *xvw_map()*.) If TRUE, this attribute will cause the object to be mapped; if FALSE, it will delay mapping of the object until so instructed.
XVW_MENU_CLIENTDATA	When *xvw_define_attribute* is used to define additional attributes of objects, this is the client data that is sent to the *set* and *get* routines.
XVW_MENU_FORM	This attribute defines the internal menuform associated with a visual object. The menuform should include selections appropriate for each attribute that can be interactively set by the user. Variable names must be equal to the resource names.
XVW_MENU_FORMFILE	This attribute defines the *.pane file which defines the internal menuform of a visual object. The *.pane file defining the menuform should include selections appropriate for each attribute that can be interactively set by the user. Variable names must be equal to the resource names.
XVW_NAME	The name used to identify the object. It is used for specification of resources in the app-defaults file; in addition, for backplanes it will appear as the name on the window dressing.
XVW_VISUAL	This is the X11 *visual* being used to display the visual object on a particular screen. Visuals are discussed in detail | in *The Xlib Programming Manual* by Adrian Nye. Note that the application should generally *not* attempt to set the visual; it should instead allow the visual object to choose the most appropriate visual.

Descriptions of General Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_BACKGROUND (background)	unsigned long	default bg pixel (XtDefaultBackground)	any valid pixel value: see Section 7.3 and 7.4 of *The XLib Programming Manual* by Adrian Nye
XVW_BACKGROUND_COLOR (backgroundColorname)	char *	default bg color	any valid color name: see /usr/lib/X11/rgb.txt or Section 7.1.1 of *The XLib Programming Manual* by Adrian Nye
XVW_BACKGROUND_PIXEL (background)	unsigned long	default bg pixel (XtDefaultBackground)	any valid pixel value: see Section 7.3 and 7.4 of *The XLib Programming Manual* by Adrian Nye

Descriptions of General Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_BACKGROUND_PIXMAP (backgroundPixmap)	Pixmap	NULL	Valid Pixmap structure
XVW_BACKGROUND_PIXMAPFILE (N/A)	char *	NULL	The full path to a valid xpm or xbm file, defining the desired pixmap. Note that the path may contain $TOOLBOX.
XVW_BORDER (border)	unsigned long	None	any valid pixel value: see Section 7.3 and 7.4 of *The XLib Programming Manual* by Adrian Nye
XVW_BORDER_COLOR (borderColorname)	char *	N/A	any valid color name: see /usr/lib/X11/rgb.txt or Section 7.1.1 of *The XLib Programming Manual* by Adrian Nye
XVW_BORDER_PIXEL (N/A)	unsigned long	None	any valid pixel value: see Section 7.3 and 7.4 of *The XLib Programming Manual* by Adrian Nye
XVW_CHAR_XSNAP (charXsnap)	float	1	values > 0.0
XVW_CHAR_YSNAP (charYsnap)	float	1	values > 0.0
XVW_COLORMAP (N/A)	Colormap	XDefaultColormap()	a valid Colormap structure.
XVW_CURSOR (N/A)	Cursor	The cursor of the parent object	Valid Cursor structure (see description).
XVW_CURSORNAME (cursorName)	char *	N/A	Valid cursorfont name (see description).
XVW_DEPTH (N/A)	int	computed according to XVW_VISUAL	the number of planes supported by the visual.

XVW_DESTROY (N/A)	void (*call-back_rou-tine)(xvob-ject, kaddr, kaddr)	NULL	callback function, in the form:
			`void callback_function`
			`xvobject object,`
			`kaddr client_data,`
			`kaddr call_data)`

XVW_FONT (N/A)	XFontStruct	default font	see description
XVW_FONTNAME (fontName)	char *	default font	see description
XVW_FOREGROUND (background)	unsigned long	default fg pixel (XtDefaultFore-ground)	any valid pixel value: see Section 7.3 and 7.4 of *The XLib Programming Manual* by Adrian Nye

Descriptions of General Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_FOREGROUND_COLOR (foregroundColorname)	char *	default fg color	any valid color name: see /usr/lib/X11/rgb.txt or Section 7.1.1 of *The XLib Programming Manual* by Adrian Nye
XVW_FOREGROUND_PIXEL (foreground)	unsigned long	default fg pixel (XtDefaultFore-ground)	any valid pixel value: see Section 7.3 and 7.4 of *The XLib Programming Manual* by Adrian Nye
XVW_MAPPED (mapped)	int	TRUE	TRUE/FALSE
XVW_MENU_CLIENTDATA (N/A)	kaddr	NULL	pointer to the client data
XVW_MENU_FORM (N/A)	kaddr	NULL	valid internal menuform structure
XVW_MENU_FORMFILE (N/A)	char *	NULL	valid path to *.pane file defining internal menuform. Use of $TOOLBOX is encouraged.
XVW_NAME (N/A)	char *	N/A	single string with no spaces
XVW_VISUAL (N/A)	Visual	XDefaultVisual(), or may be deter-mined according to the visual object being used and the screen on which it is being dis-played	a valid Visual structure

C. Toplevel (Shell) Objects

An X11 *client*, or application program must always create one or more *toplevel* windows which are children of the root window. It is the use of a toplevel window that allows an application to work with a window manager; the window manager affects and interacts with these toplevel windows. If an X application brings up two separate displays, each of which is controllable with the window manager, this generally implies that the application has two toplevel windows.

The X Toolkit provides a few different kinds of toplevel widgets. VisiQuest 2001 uses *application shell widgets* and *transient shell widgets*; these widgets are presented as GUI objects. We will not go into a long explanation of the differences between application shell widgets and transient shell widgets here; more information on shell widgets can be found in *The X Toolkit Intrinsics Reference Manual* by Adrian Nye and Tim O'Reilly. For now, the main displays for VisiQuest 2001 applications will be application shell widgets, while any short-lived popup windows, such as error messages, will be transient shell widgets.

Toplevels cannot display GUI or visual objects themselves; rather, they have a single child, which in VisiQuest 2001 applications is usually a VisiQuest 2001 Manager widget. The VisiQuest 2001 Manager widget will act as the backplane, or parent, for any GUI or visual objects which are to be displayed by the application.

For a VisiQuest 2001 application with a GUI specified by a UIS file, the *xvforms* library will automatically create a toplevel containing a single VisiQuest 2001 Manager widget child. The *xvforms* library then uses the this VisiQuest 2001 Manager widget as a parent for all the other GUI objects that are created as part of the graphical user interface of the application.

It is rarely necessary to add code that will create a toplevel in a classic xvroutine (an xvroutine having a GUI specified with a *.form file). Sometimes it is desirable to add code that will create a toplevel in a hybrid xvroutine (an xvroutine without a GUI specified by a *.form file). Since a hybrid xvroutine does not use the *xvforms* library to create a GUI, any image or graphical display must be done directly from the hybrid routine; the first step in this process is to create a toplevel widget with a single VisiQuest 2001 Manager as the child. The ubiquitous call to *xvw_create_manager()* will create a default toplevel for you if the parent argument is specified as NULL; however, it is sometimes preferable to create the toplevel explicitly. Toplevels may be created explicitly using the following two functions:

C.1. xvw_create_application_shell() — *create an application shell object*

Synopsis

```
xvobject xvw_create_application_shell(

    char    *name,
    Display *display,
    Screen  *screen)
```

Input Arguments

`name`
: name for the application shell object

`display`
: the X Display structure; pass NULL to use the default display

`screen`
: the X Screen structure; pass NULL to use the default screen

Returns

returns the application shell object on success, NULL on failure

Description

Creates an application shell object to serve as a toplevel for an application; the toplevel serves as a mediating device between the application and the window manager. This call allows the application to have several independant windows, which is the case with xvroutines having multiple subforms. The routine also supports the creation of toplevels on potentially different screens, which applies when a

VisiQuest application distributes its user interface with the concert program. Both the display and screen are optional arguments; if either of these are set to NULL, then the default display and default screen will be used.

C.2. xvw_create_transient_shell() — *create a transient shell object*

Synopsis

```
xvobject xvw_create_transient_shell(

    char    *name,
    Display *display,
    Screen  *screen)
```

Input Arguments

`name`
: name for the transient shell object

`display`
: the X Display structure; pass NULL to use the default display

`screen`
: the X Screen structure; pass NULL to use the default screen

Returns

the transient shell object on success, NULL on failure

Description

Creates an transient shell object to serve as the toplevel for a popup object. Both the display and the screen are optional arguments; if either is set to NULL, then the default display and default screen are used.

C.3. Attributes of the Shell Object

Summary of Toplevel (Shell) Attributes	
Attribute	**Description**
XVW_SHELL_ICONIFY	Iconify or deiconify the toplevel object.

Summary of Toplevel (Shell) Attributes	
Attribute	**Description**
XVW_SHELL_ICON_MASK	Pixmaps are defined by a rectangular grid of values. For implementing non-rectangular pixmaps, a bitmap is used to indicate which portions of the rectangular pixmap should appear, and which portions should not be displayed. This bitmap is referred to as a *mask*. The bitmap must be defined in an xbm file, and the bitmap should be of the same size as the pixmap being displayed. Anywhere a 1 appears in the bitmap, the value of the pixmap at that point is displayed; anywhere a 0 appears in the bitmap, the value of the pixmap at that point will not be displayed. Thus, a circular pixmap can be implemented with a pixmap/bitmap mask pair: the pixmap will define the picture in the circle, while the bitmap mask has all 1's within the boundaries of the circle, and all 0's outside the circle bounds. This is the mask used with the icon pixmap, if any. Candidates for the bitmap may be created with *XCreateBitmapFromData()*; see *The Xlib Reference Manual* by O'Reilly and Associates. Note that this attribute is mutually exclusive with XVW_SHELL_ICON_MASKFILE; specify one or the other, not both.
XVW_SHELL_ICON_MASKFILE	This is the file defining the bitmap mask used with non-rectangular pixmaps; see XVW_SHELL_ICON_MASK for more details.
XVW_SHELL_ICON_NAME	This is the name that appears on the icon associated with the shell when the application is iconified.
XVW_SHELL_ICON_PIXMAP	This is the pixmap that appears on the icon when the application is iconified. Candidates for the value of this attribute may be created with the use of *XCreatePixmap()*; see *The Xlib Reference Manual* by O'Reilly and Associates. Note that this attribute is mutually exclusive with XVW_SHELL_ICON_PIXMAPFILE; specify one or the other, not both. Note that setting this attribute will over-ride use of the icon name on the icon, as specified by the value of XVW_SHELL_ICON_NAME.
XVW_SHELL_ICON_PIXMAPFILE	The file defining the pixmap that appears on the icon when the application is iconified.
XVW_SHELL_ICON_WINDOW	The window that is displayed when the application is iconified.
XVW_SHELL_INITIAL_STATE	The window's initial state. This specifies how the shell's toplevel should be created, whether in a normal, iconic, or withdrawn state.
XVW_SHELL_INPUT	If set to TRUE, the window manager will set the keyboard focus to this application or not, according to its pointer- following or click-to-type model of keyboard input. If set to FALSE, the window manager will not set the keyboard focus to this application. For more details, see *The X Toolkit Intrinsics Programming Manual* by Adrian Nye and Tim O'Reilly, section 10.1.4.
XVW_SHELL_RESIZE	Specifies whether or not the shell will request size change from the window manager.
XVW_SHELL_TITLE	This is the title that appears on the window dressing of the toplevel object.

Summary of Toplevel (Shell) Attributes	
Attribute	**Description**
XVW_SHELL_WIN_GRAVITY	The window gravity controls the repositioning of subwindows when a parent window is resized. See Section 4.3.4. of *The Xlib Programming Manual* by Adrian Nye for more information.
XVW_SHELL_X	For automatic placement of the toplevel object, specify the (x,y) location on the screen where the toplevel object is to be automatically mapped. XVW_SHELL_X specifies the X location for automatic placement.
XVW_SHELL_Y	For automatic placement of the toplevel object, specify the (x,y) location on the screen where the toplevel object is to be automatically mapped. XVW_SHELL_Y specifies the Y location for automatic placement.

Descriptions of Toplevel (Shell) Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_SHELL_ICONIFY (N/A)	int	N/A	TRUE/FALSE
XVW_SHELL_ICON_MASK (iconMask)	Pixmap	NULL	Valid Pixmap structure
XVW_SHELL_ICON_MASKFILE (N/A)	char *	NULL	The full path to the xbm file defining the mask; Note that the path may contain $TOOLBOX.
XVW_SHELL_ICON_NAME (iconName)	char *	The application name or the value of XVW_SHELL_TITLE; the default is set by the window manager.	any printable text
XVW_SHELL_ICON_PIXMAP (iconPixmap)	Pixmap	NULL	Valid Pixmap structure
XVW_SHELL_ICON_PIXMAPFILE (N/A)	char *	NULL	The full path to a valid xpm or xbm file, defining the desired pixmap. Note that the path may contain $TOOLBOX.
XVW_SHELL_ICON_WINDOW (N/A)	Window	Defined by window manager	any valid Window
XVW_SHELL_INITIAL_STATE (initialState)	int	NormalState	WithdrawnState, NormalState, IconicState
XVW_SHELL_INPUT (shellInput)	int	TRUE	TRUE/FALSE
XVW_SHELL_RESIZE (shellResize)	int	TRUE	TRUE/FALSE

Descriptions of Toplevel (Shell) Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_SHELL_TITLE (shellTitle)	char *	None (if shell is created with *xvf_create_form()*, it will be set by the *xvforms* library to the title of the Form given on the [-F] UIS line for GUI's with master forms, or to the title of the Subform given on the [-M] UIS line for subforms).	any printable text
XVW_SHELL_WIN_GRAVITY (winGravity)	int	NorthWestGravity	NorthWestGravity, NorthGravity, NorthEastGravity, WestGravity, CenterGravity, EastGravity, SouthWestGravity, SouthGravity, or SouthEastGravity
XVW_SHELL_X (shellX)	int	N/A	0 <= value <= screen width
XVW_SHELL_Y (shellY)	int	N/A	0 <= value <= screen height

D. Setting And Getting Attributes

The characteristics of GUI and visual objects that can be specified by the application are called GUI and visual object *attributes*. GUI and visual object *resources* are pairs of names and values that indicate the current setting of object attributes. Attributes of GUI and visual objects are defined by:

1) The object itself (see "GUI Objects")
2) The constraint resources of its VisiQuest 2001 Manager parent, if applicable (see "The VisiQuest 2001 Manager Widget").
3) The general attributes that are common to all GUI and visual objects (see "General Attributes of GUI & Visual Objects")

Attributes of GUI and visual objects can be set and retrieved one at a time using *xvw_set_attribute()* and *xvw_get_attribute()*. These routines have three fixed arguments: the object, followed by the (attribute/value)

resource pair. The definitions are as follows.

D.1. xvw_set_attribute() — *set a single attribute on an object*

Synopsis

```
xvw_set_attribute(

    xvobject object,
    char     *attribute,
    data     value)
```

Input Arguments

`object`
: the object for which to set the attribute

`attribute`
: the attribute name

`value`
: the attribute value

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

sets a single attribute for a visual or GUI object

Restrictions

Restrictions on data or input as applicable

D.2. xvw_get_attribute() — *get a single attribute of an object*

Synopsis

```
xvw_get_attribute(

    xvobject object,
    char     *attribute,
    data     *value)
```

Input Arguments

`object`
the object for which to get the attribute

`attribute`
the attribute name

Output Arguments

`value`
returns the current attribute value

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

gets a single attribute for a visual or GUI object

Restrictions

Restrictions on data or input as applicable

Alternatively, a group of attributes can be set or retrieved at the same time using the *xvw_set_attributes()* and *xvw_get_attributes()* routines. These variable argument routines have a fixed first argument (the object), followed by pairs of attributes and values; finally, both routines *must* have NULL as their last argument to indicate the end of the attribute/value pairs. Neglecting to provide NULL as the last argument to either routine will most likely result in a segmentation fault. The definitions of these two commonly-used routines are as follows.

D.3. xvw_set_attributes() — *set attributes on an object (variable argument list)*

Synopsis

```
int xvw_set_attributes(

  xvobject object,
  kvalist)
```

Input Arguments

`object`
the object on which to set the attributes

`kvalist`
variable argument list, must be in the form:

```
XVW_ATTRIBUTE_1,   value1,
XVW_ATTRIBUTE_2,   value2,
```

```
XVW_ATTRIBUTE_3,    value3,
:
:
XVW_ATTRIBUTE_N,    valueN,
NULL
```

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

Sets a variable number of attributes associated with a GUI or visual object.

D.4. xvw_get_attributes() — *get attributes from an object (variable argument list)*

Synopsis

```
int xvw_get_attributes(

  xvobject object,
  kvalist)
```

Input Arguments

`object`

the object for which to get the attributes

`kvalist`

variable argument list, must be in the form:

```
XVW_ATTRIBUTE_1,    pointer1,
XVW_ATTRIBUTE_2,    pointer2,
XVW_ATTRIBUTE_3,    pointer3,
:
:
XVW_ATTRIBUTE_N,    pointerN,
NULL
```

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

Gets a variable number of attributes associated with a GUI or visual object.

There are some special rules for the acquiring of strings that you should know. The protocol followed by the *xvwidget* library with respect to obtaining string attributes is the same as that followed by the X Toolkit. Strings are passed by address, and the string returned is simply the *address of the string as it appears internally*. That is, the string returned to you is NOT allocated for you, and you may NOT change the value of the string in any way; it is for inspection purposes only. You do not need to initialize the string, and you should NEVER free the string. If you intend to change the string and re-set it, you must make a copy of the string, change the copy, call the appropriate *set* routine, and free the copy of the string when it is no longer used. Alternatively, you may copy the string into a buffer and change it without doing an allocation or a free. The following code segments demonstrate the proper use of a string attribute:

```
/*
 * ------------------------------------------------------------
 *  example 1 changing the attribute using a local allocated string
 */
char *return_string, *string_value;

/* get the value of the string */
xvw_get_attribute(object, XVW_SOME_STRING_ATTRIBUTE, &return_string);

/* copy the string into a local string variable */
string_value = kstring_copy(return_string, NULL);

/* inspect string_value, change string_value as desired */

/* set the attribute to the new value of the string */
xvw_set_attribute(object, XVW_SOME_STRING_ATTRIBUTE, string_value);

/* free the local string variable when it is no longer used */
kfree(string_value);

/*
 * ------------------------------------------------------------
 *  example 2 changing the attribute using a local static buffer
 */
char *return_string;
char buffer[KLENGTH];

/* get the value of the string */
xvw_get_attribute(object, XVW_SOME_STRING_ATTRIBUTE, &return_string);

/* append "_name" onto end of string */
sprintf(buffer, "%s_name", return_string);

/* set the attribute to the new value of the string */
xvw_set_attribute(object, XVW_SOME_STRING_ATTRIBUTE, buffer);
```

E. The VisiQuest 2001 Manager Object

The VisiQuest 2001 *manager object* is a central element of the VisiQuest 2001 Widget Set. It serves multiple functions in the creation, layout, and maintenance of the GUI and visual objects that are created in a VisiQuest 2001 application that supports visual display.

To begin, some distinctions must be made in terminology used when referring to the manager object. There are actually *two* implementations of the VisiQuest 2001 Manager: the manager *widget* and the manager *gadget*. When we refer to other visual and GUI objects in GUI and Visualization Services, we make no distinction between widgets and gadgets; they are all referred to as "objects." For example, an annotation such as a circle object is a *gadget*; it has no window created specifically for it, but rather it "borrows" the window of its parent. A button, on the other hand, is a *widget*; it has a dedicated window of its own. In spite of their differences, both the circle and the button are called *objects*.

The manager object is unique in GUI and Visualization Services, in that it is implemented in both ways. The manager *widget* is used to subclass other visual and GUI objects that are also *widgets*, such as the image object, the area object, and the zoom object. The manager *gadget* is used to subclass other visual objects that are also *gadgets*, such as axis objects, plot objects, and the various annotation objects. Manager *widgets* are created explicitly and used often. In contrast, you cannot create a manager *gadget*; the only purpose of the manager gadget is to act as a superclass for the visual objects that are implemented as gadgets.

Throughout the documentation, the Manager widget is often referred to as a manager *object*, in order to maintain consistency with the use of the term "object" when referring to all the other widgets & gadgets provided by GUI and Visualization Services. On the rare occasions when the manager *gadget* is meant specifically, it will be referred to as the "manager gadget."

Since it has its own window, the manager object can serve as a parent, or *backplane*, for other GUI and visual objects; this is not true of the manager gadget. A *constraint* widget, the manager object manages the layout of its children and allows different layout rules to be provided for each child. After the toplevel object for an application is created, its child should usually be a Manager object. [1] The manager object, in turn, can be used as a parent for any number of objects, or other manager objects. The Manager object provides convenience layout attributes and augmented control attributes. In addition, it has a variety of attributes to support the direct manipulation of its children.

An important function of the manager object is to support the direct manipulation of child objects. Because they are created with a manager object as a parent, GUI and visual objects can be moved and resized interactively by the user. An object can be *selected* with a *meta-click* - that is, a mouse click on the object using the first mouse button while the *meta* key is held down (the key that functions as *meta* will depend on the particular key mapping of your keyboard -- most common are shift, control, compose, and alt). Once an object is selected, it can be resized by holding the first mouse button down on an edge and "dragging" the edge to the desired size, or it can be moved by holding the mouse button down in the middle of the object and dragging it to the desired location.

In the same spirit as interactive sizing and positioning, another crucial function of the manager object is to support the creation and display of *internal menuforms* associated with child objects. An internal menuform is an independent GUI that is associated with a particular object, and presents the user with a way of setting the attributes of that particular object. After selecting the object with a meta-click, the user can bring up the internal menuform of the object by clicking on the object with the second mouse button, and use the menuform to set attributes of the object as desired.

The library responsible for the creation of the GUI or visual object also controls the operation of the internal menuforms that are associated with the objects; however, it is the manager object that associates the internal menuform with its object and brings up the menuform when it is requested by the user. Internal menuforms

[1] Exceptions to this are when a constraint object subclassed from the Manager is used instead, such as a RowCol object.

are defined and supported for all GUI and visual objects available in the *xvwidgets*, *xvforms*, *xvisual*, and *xvlang* libraries. Internal menuforms provide a powerful mechanism for allowing GUI and visual objects to directly handle user modification of their attributes; the application program is freed from the need to implement any of the native capabilities of the object.

In summary, the manager object is used as a parent for both GUI and visual objects, and supports a variety of resources that control the layout of its children. It supports the direct manipulation of GUI and visual objects, and allows them to bring up internal menuforms that can be used to set their attributes.

E.1. xvw_create_manager() — *create a VisiQuest Manager object*

Synopsis

```
xvobject xvw_create_manager(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`

parent of the Manager object: a toplevel object, another Manager object, or NULL. If NULL is provided, a default toplevel object will be created automatically to serve as a parent for the manager.

`name`

a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The Manager object on success, NULL on failure

Description

Creates a VisiQuest Manager object to serve as a backplane (parent) for visual objects and/or GUI objects . The following widget is the base widget used by all graphical user interface programs. This manager plays an integral role in managing all VisiQuest related applications. It is the glue by which several important VisiQuest features are implemented.

The features that this widget provide are:

1) common manager widget to make implementing multiple widgets set possible.

2) the ability of providing an interactive editing capibility to every application.

3) with the advent of an object (gadget) based annotations capibility. We want any application to be able to advantage of this feature with little or no work. (this also enforces a common interface).

First off by having a common widget "manager" we effectively make the implementation of multiple widget sets possible. The idea is that most widgets sets such as

Athena, Motif, Open Look (Olit)

have similar objects such as buttons, labels, lists, etc. But the manager widgets are not similar. They all have a different layout philsophy and terminolgy thus making it difficult to write transparent code for. Also, since we originally wrote Khoros 1.0 with the MIT Form Widget in mind, this part of the system is extremely similar in terminology.

E.2. Attributes of the VisiQuest 2001 Manager Object

The VisiQuest 2001 Manager widget is a *constraint* widget; that is, it manages the layout of its children *and* allows the application to provide layout information for each child. Thus, you may provide different rules for how each child will be laid out, if applicable. This section details a number of constraint resources that are provided by the VisiQuest 2001 Manager widget. Remember that these resources may be set on any GUI or visual object, *provided that the object in question has a Manager object for a parent.*

E.2.1. Relative Layout Attributes

The following attributes are used to specify the relative location of a object. When another (previously created) object is specified, the object is relative to the location of the other object. Explicitly setting one of these attributes to NULL will result in a location relative to the edges of the parent. Note that default values are not NULL but undefined.

Summary of Relative Layout Attributes

Attribute	Description
XVW_ABOVE	The object having the attribute set will appear above the object specified. Setting the attribute to NULL implies that the object will be above *nothing*; ie, it will appear at the bottom of the parent.
XVW_BELOW	The object having the attribute set will appear below the object specified. Setting the attribute to NULL implies that the object will be below *nothing*; ie, it will appear at the top of the parent.
XVW_LEFT_OF	The object having the attribute set will appear to the left of the object specified. Setting the attribute to NULL implies that the object will be left of *nothing*; ie, it will appear at the right edge of the parent.
XVW_RIGHT_OF	The object having the attribute set will appear to the right of the object specified. Setting the attribute to NULL implies that the object will be right of *nothing*; ie, it will appear at the left edge of the parent.

Descriptions of Relative Layout Attributes

Attribute (Resource Name)	Type	Default	Legal Values
XVW_ABOVE (N/A)	xvobject	MANAGER_UNDEFINED	NULL, or sibling visual object
XVW_BELOW (N/A)	xvobject	MANAGER_UNDEFINED	NULL, or sibling visual object

Descriptions of Relative Layout Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_LEFT_OF (N/A)	xvobject	MANAGER_UNDEFINED	NULL, or sibling visual object
XVW_RIGHT_OF (N/A)	xvobject	MANAGER_UNDEFINED	NULL, or sibling visual object

The *relative layout attributes* can be used in combination to achieve various effects. For example, suppose a object is specified to be XVW_ABOVE NULL and XVW_BELOW NULL. This causes it to "pull" with equal weight to the bottom and top edges of the parent, resulting in a location in the vertical middle of the parent. The following table shows some effects that can be achieved by setting various combinations to NULL. Use of a "-" in the table indicates that the attribute is not specified (left undefined).

Configurations Specified using Relative Layout WRT Parent Using NULL				
Desired Location	*XVW_ABOVE*	*XVW_BELOW*	*XVW_RIGHT_OF*	*XVW*
_LEFT_OF				
Object in upper left hand corner.	-	NULL	NULL	-
Object in upper right hand corner.	-	NULL	-	NULL
Object in lower right hand corner.	NULL	-	-	NULL
Object in lower left hand corner.	NULL	-	NULL	-
Object on left edge of parent, in vertical middle	NULL	NULL	NULL	-
Object on right edge of parent, in vertical middle	NULL	NULL	-	NULL
Object on top edge of parent, in horizontal middle	NULL	-	NULL	NULL
Object on bottom edge of parent, in horizontal middle	-	NULL	NULL	NULL
Object in exact center of parent.	NULL	NULL	NULL	NULL

E.2.2. Pixel Geometry Bounds Attributes

The following attributes are used to specify bounds on object geometry in pixels. Both take integer values, and will take precedence over any counterparts specified in character widths (see "Character Geometry Bounds").

Summary of Attributes That Control Maximum/Minimum Sizing	
Attribute	**Description**
XVW_MAXIMUM_HEIGHT	Maximum width of the object in pixels.
XVW_MAXIMUM_WIDTH	Maximum width of the object in pixels.
XVW_MINIMUM_HEIGHT	Minimum height of the object in pixels.
XVW_MINIMUM_WIDTH	Minimum height of the object in pixels.

Descriptions of Attributes That Control Maximum/Minimum Sizing			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_MAXIMUM_HEIGHT (maximumHeight)	int	KMANAGER_UNDEFINED	value > 0
XVW_MAXIMUM_WIDTH (maximumWidth)	int	KMANAGER_UNDEFINED	value > 0
XVW_MINIMUM_HEIGHT (minimumHeight)	int	1	value > 0
XVW_MINIMUM_WIDTH (minimumWidth)	int	1	value > 0

E.2.3. Preferred Sizing Attributes

Preferred sizing is used to tell the manager the initial size from which to do its own geometry layout, before doing geometry layout of its children. For example, suppose the preferred width and height are both set to 512. The manager object will begin by sizing itself at (512x512). However, if it requires a larger size than (512x512) in order to accomodate its children, it will grow as necessary. Thus, preferred sizing can be thought of as a geometry "hint;" it specifies a reasonable starting size for the manager, but the actual size of the manager when it is mapped may be larger if necessary.

Summary of Preferred Geometry Attributes	
Attribute	**Description**
XVW_PREFERRED_HEIGHT	The preferred height of the manager object.
XVW_PREFERRED_WIDTH	The preferred width of the manager object.

Descriptions of Preferred Geometry Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_PREFERRED_HEIGHT (preferredHeight)	int	undefined	value > 0
XVW_PREFERRED_WIDTH (preferredWidth)	int	undefined	value > 0

E.2.4. Pixel Spacing Attributes

Pixel spacing can be used in conjunction with relative layout in order to achieve the desired effect on a display. All attributes are set with integer values.

Summary of Attributes That Control Spacing	
Attribute	**Description**
XVW_BUFFER_DIST	This is the buffer distance in pixels around the edge of the Manager. It represents the minimum number of pixels that will appear between a child and the nearest edge of its parent.
XVW_DEF_HORIZ_DIST	This attribute is used only with *constraint* objects that will be laying out children, such as manager objects, rowcol objects, viewport objects, and so on. It sets the default horizontal distance, in pixels, that children will be spaced from one another when doing relative layout. Suppose you have a manager object containing many buttons, each of which is right of the last. If you wanted each button to appear 5 pixels to the right of the one before it, you could set XVW_HORIZ_DIST to 5 on each of the children. An easier approach, however, would be to simply set XVW_DEF_HORIZ_DIST to 5 once, on the manager object parent. Note that the value specified by XVW_DEF_HORIZ_DIST will be over-ridden by individual horizontal distances specified by any of the children using XVW_HORIZ_DIST.
XVW_DEF_VERT_DIST	This attribute is used only with *constraint* objects that will be laying out children, such as manager objects, rowcol objects, viewport objects, and so on. It sets the default vertical distance, in pixels, that children will be spaced from one another when doing relative layout. Suppose you have a manager object containing many buttons, each of which is below the last. If you wanted each button to appear 7 pixels below the one above it, you could set XVW_VERT_DIST to 7 on each of the children. An easier approach, however, would be to simply set XVW_DEF_VERT_DIST to 7 once, on the manager object parent. Note that the value specified by XVW_DEF_VERT_DIST will be over-ridden by individual vertical distances specified by any of the children using XVW_VERT_DIST.

Summary of Attributes That Control Spacing	
Attribute	**Description**
XVW_HORIZ_DIST	Horizontal distance in pixels (to the right) from the location at which the object would otherwise be placed. This is most often used in conjunction with relative layout. For example, you might use the XVW_RIGHT_OF attribute to make an object appear to the right of a second object; XVW_HORIZ_DIST can then be used to specify *how many* pixels to the right the first object should be spaced from the second.
XVW_VERT_DIST	Vertical distance in pixels (down) from the location at which the object would otherwise be placed. This is most often used in conjunction with relative layout. For example, you might use the XVW_BELOW attribute to make an object appear below a second object; XVW_VERT_DIST can then be used to specify *how many* pixels below the second object the first object should appear.

Descriptions of Attributes That Control Spacing			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_BUFFER_DIST (bufferDist)	int	3	values >= 0
XVW_DEF_HORIZ_DIST (defHorizDist)	int	3	values >= 0
XVW_DEF_VERT_DIST (defVertDist)	int	3	values >= 0
XVW_HORIZ_DIST (horizDist)	int	0	values >= 0
XVW_VERT_DIST (vertDist)	int	0	values >= 0

E.2.5. Tacking Attributes

Tacking is a technique that is used to easily control size and positioning of GUI and visual objects. Tacking causes an edge of the child object to be "attached" to the nearest edge of its parent or closest neighbor, so that regardless of how the object or its parent are moved or resized, the tacked edge is always a fixed distance from the edge to which it is tacked. The child will stretch or shrink as necessary to preserve the small fixed distance between its tacked edge and the edge of its parent (or neighbor).

The fixed distance that is maintained between the tacked edge and the edge of its parent (or neighbor) is determined by the values of the pixel spacing attributes (XVW_HORIZ_DIST, XVW_VERT_DIST, XVW_DEF_HORIZ_DIST, and XVW_DEF_VERT_DIST)

An edge can be tacked to the right, to the left, to the top, or to the bottom. In order to achieve all possible combinations, values for tacking can be OR'd together to indicate horizontal tacking, vertical tacking, tacking on all edges, or any other desired combination.

Summary of Tacking Attributes	
Attribute	**Description**
XVW_TACK_EDGE	Tacks the edge(s) of the object to the edge(s) of its nearest neighbor or parent. This attribute takes a mask. These masks may be OR'ed together in order to achieve the desired tacking setup. Values accepted include: KMANAGER_TACK_LEFT: !Set to NULL: Tack the left edge of the object ! to the left edge of its parent !Set to non-NULL: Tack the left edge of the object ! to the right edge of its nearest neighbor ! on the left. KMANAGER_TACK_RIGHT: !Set to NULL: Tack the right edge of the object ! to the right edge of its parent !Set to non-NULL: Tack the right edge of the object ! to the left edge of its nearest neighbor ! on the right. KMANAGER_TACK_TOP: !Set to NULL: Tack the top edge of the object ! to the top edge of its parent !Set to non-NULL: Tack the top edge of the object ! to the bottom edge of the nearest neighbor ! above it. KMANAGER_TACK_BOTTOM: !Set to NULL: Tack the bottom edge of the object ! to the bottom edge of its parent !Set to non-NULL: Tack the bottom edge of the object ! to the top edge of the nearest neighbor ! below it. KMANAGER_TACK_VERT: Tack the left and right edges of the object at the same time; note that this is the same as OR'ing together KMANAGER_TACK_LEFT and MANAGER_TACK_RIGHT, and passing the result. KMANAGER_TACK_HORIZ: Tack the top and bottom edges of the object at the same time; note that this is the same as OR'ing together KMANAGER_TACK_TOP KMANAGER_TACK_BOTTOM, and passing the result. KMANAGER_TACK_ALL: Tack all four edges of the object at the same time; note that this is the same as OR'ing together KMANAGER_TACK_LEFT, KMANAGER_TACK_RIGHT, KMANAGER_TACK_TOP, and KMANAGER_TACK_BOTTOM, and passing the result. KMANAGER_TACK_NONE: Disable tacking

Descriptions of Tacking Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_TACK_EDGE (N/A)	int	KMANAGER_TACK_NONE	KMANAGER_TACK_NONE KMANAGER_TACK_LEFT KMANAGER_TACK_RIGHT KMANAGER_TACK_TOP KMANAGER_TACK_BOTTOM KMANAGER_TACK_VERT KMANAGER_TACK_HORIZ KMANAGER_TACK_ALL

E.2.6. Attributes That Control Direct Manipulation of Children

These resources are used to control the type and amount of direct manipulation that is enabled on a particular GUI or visual object, if any.

Summary of Attributes that Enable Direct Manipulation of Children	
Attribute	**Description**
XVW_CANVAS_GRID	This attribute indicates when a grid should be displayed on the canvas. KMANAGER_GRID_OFF specifies that the grid is always off. KMANAGER_GRID_ON specifies that the grid is always on. KMANAGER_GRID_EDIT specifies that the grid is only on when the canvas is in "edit mode". The canvas can be put in edit mode by the user, or through the application by setting the manager object attribute XVW_EDIT_MODE_ON to TRUE. KMANAGER_GRID_SELECT specifies that the grid is only displayed when a child of the canvas has been selected by the user, or through the application by setting the manager object attribute XVW_SELECT_ADD to TRUE.
XVW_CANVAS_GRIDSIZE	The dimensions of the grid, given in pixels.
XVW_CHILDREN	This *read only* attribute allows you to obtain an array of the *Widgets* representing the children of the specified object. The fact that this attribute returns Widgets rather than xvwidgets represents an inconsistency; however, due to internal limitations, the children must be returned as Widgets at this time. Note that a Widget may always be converted to an xvobject with the use of *xvw_object()*.

XVW_EDIT_MODE_CALLBACK	If desired, *xvw_add_callback()* may be used to install a callback on the manager object which will be fired when the user puts it into or takes it out of edit mode. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(manager_object, XVW_EDIT_MODE_CALLBACK,`
	`callback_function, client_data);`
	Note that in order to allow the user to select the manager object, its XVW_SELECTABLE attribute must be TRUE (the default).

Summary of Attributes that Enable Direct Manipulation of Children	
Attribute	**Description**
XVW_EDIT_MODE_ON	This attribute allows the application to control whether or not a visual object is in *edit mode*. When the visual object is in "edit mode", it will usually display a grid where each square is 1/2 character width by 1/2 character height. Once an object is in edit mode, it will allow the user to select, resize, and move its children (provided that the children have XVW_SELECTABLE and XVW_RESIZABLE set to TRUE, respectively. Note that the user can put a visual object in edit mode by doing a "meta-click", ie, by holding down the "meta" key (the key which acts as "meta" on a keyboard will depend on the keyboard mapping -- most common are "Alt", "Compose", "Ctrl", and "Shift"), and simultaneously clicking the left mouse button on the visual object.

XVW_GEOMETRY_CALLBACK	If desired, *xvw_add_callback()* may be used to install a callback on the visual object which will be fired when the user resizes or moves the object. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(visualobj, XVW_GEOMETRY_CALLBACK,`
	`callback_function, client_data);`
	Note that in order to allow the user to resize the object, the XVW_RESIZABLE attribute must be set to TRUE; to be selected and moved, the XVW_SELECTABLE attribute must be set to TRUE. Inside the callback, the new width, height, x and y position of the visual object may be obtained using pixel geometry attributes or character geometry attributes.

XVW_GROUP_ADD	This *action attribute* causes the specified object to be added into the child's *group list*. The object will then be selected when the child is selected.
XVW_GROUP_DELETE	This *action attribute* causes the specified object to be deleted from the child's *group list*. The object will then no longer be selected when the child is selected.
XVW_GROUP_DELETE_ALL	This *action attribute* causes the child's entire *group list* to be deleted. All the previously grouped siblings are deleted from the *group list* of the object of which the attribute is set.
XVW_MENUABLE	If TRUE, will allow the user to bring up an internal menuform for the object. Note that the *.pane file describing the internal menuform for the object must be specified using XVW_MENU_FORM.
XVW_NUM_CHILDREN	This *read only* attribute is used in conjunction with XVW_CHILDREN; it returns the number of children of the object specified.
XVW_NUM_SELECTIONS	This *read only* attribute is used in conjunction with XVW_SELECTIONS; it returns the number of selected children in the object specified.

Summary of Attributes that Enable Direct Manipulation of Children	
Attribute	**Description**
XVW_REFRESH_CHILDREN_ON_EDIT	When set to TRUE, the manager object will constantly refresh its children, even when they are interactively being moved or resized (edited) by the user. When set to FALSE, the the user has finished editing (moving or resizing) the child.
XVW_RESIZABLE	If TRUE, this attribute will allow the user to interactively resize the object after selecting it, when the parent is in *edit mode* .
XVW_SELECTABLE	If TRUE, will allow the user to select the object when the parent is in *edit mode*. When an item is selected, it will be "bracketed". A *select list* is kept; this is a list of all the children that are selected at any given time. The application may inquire which children are selected at any given time by using the XVW_XVW_SELECTIONS attribute. Note that the manager object will allow selected objects to be moved by the user.
XVW_SELECTIONS	This *read only* attribute only applies to *constraint* objects, such as manager objects, canvas objects, rowcol objects, and so on, that support layout and direct manipulation of children. If the XVW_SELECTABLE attribute is TRUE, the user will be allowed to interactively select child objects of the specified object. Selected objects will be "bracketed" on all four corners to indicate that they are currently selected. The XVW_SELECTIONS attribute may be used to obtain an array of the currently selected child objects of the object specified.
XVW_SELECT_ADD	This *action attribute* causes the specified object to be selected, and adds it to the *select list* of the parent.
XVW_SELECT_ADD_ALL	This *action attribute* takes *all* children of the object on which the attribute is set, and selects them. All the children are added to the *select list* of the object on which the attribute is set.

XVW_SELECT_CALLBACK	If desired, *xvw_add_callback()* may be used to install a callback on the manager object which will be fired when the user selects any of its children. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(manager_object, XVW_SELECT_CALLBACK,`
	`callback_function, client_data);`
	Note that in order to allow the user to select children of the manager object, the XVW_SELECTABLE attribute must be set to TRUE on the manager object.

XVW_SELECT_DELETE	This *action attribute* causes the specified object to be un-selected, and deletes it from the *select list* of the parent.
XVW_SELECT_DELETE_ALL	This *action attribute* takes *all selected children* of the object on which the attribute is set, and un-selects them. All the previously selected children are deleted from the *select list* of the object of which the attribute is set.

Summary of Attributes that Enable Direct Manipulation of Children	
Attribute	**Description**
XVW_SELECT_REPLACE	This *action attribute* allows the application to un-select any currently selected children of the parent of the specified object, and instead force the specified object to be the one that is selected. This causes the previous *select list* to be replaced with a list of size 1, containing only the object specified.
XVW_SNAP_ON	This attribute only applies to *constraint* objects, such as manager objects, canvas objects, rowcol objects, and so on, that support layout and direct manipulation of children. With interactive movement of selected child objects, the child objects can be made to "snap" to the new position, aiding the user in more precise interactive layout of objects. This attribute specifies whether the XVW_XSNAP and XVW_YSNAP should be used when directly manipulating children.

XVW_UNSELECT_CALLBACK	If desired, *xvw_add_callback()* may be used to install a callback on the manager object which will be fired when the user unselects any of its children. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(manager_object, XVW_UNSELECT_CALLBACK,`
	`callback_function, client_data);`
	Note that in order to allow the user to unselect children of the manager object, the XVW_SELECTABLE attribute must be set to TRUE on the manager object.

XVW_XSNAP	This attribute only applies to *constraint* objects, such as manager objects, canvas objects, rowcol objects, and so on, that support layout and direct manipulation of children. With interactive movement of selected child objects, the child objects can be made to "snap" to the new position, aiding the user in more precise interactive layout of objects. This attribute specifies, in pixels, the horizontal increments of the implied grid (visible or not) to which children will "snap". For example, if XVW_XSNAP is set to 15, then objects will "snap" to positions in 15-pixel increments.
XVW_YSNAP	This attribute only applies to *constraint* objects, such as manager objects, canvas objects, rowcol objects, and so on, that support layout and direct manipulation of children. With interactive movement of selected child objects, the child objects can be made to "snap" to the new position, aiding the user in more precise interactive layout of objects. This attribute specifies, in pixels, the vertical increments of the implied grid (visible or not) to which children will "snap". For example, if XVW_YSNAP is set to 20, then objects will "snap" to positions in 20-pixel increments.

Descriptions of Attributes That Enable Direct Manipulation of Children			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_CANVAS_GRID (canvasGrid)	int	KMANAGER_GRID_SELECT	KMANAGER_GRID_OFF KMANAGER_GRID_ON KMANAGER_GRID_EDIT KMANAGER_GRID_SELECT
XVW_CANVAS_GRIDSIZE (canvasGridsize)	int	15	value > 0
XVW_CHILDREN (N/A)	Widget *	N/A	Widget array of the children of the specified object.
XVW_EDIT_MODE_CALLBACK (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_EDIT_MODE_ON (editModeOn)	int	FALSE	TRUE/FALSE
XVW_GEOMETRY_CALLBACK (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr);	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_GROUP_ADD (N/A)	xvobject	N/A	Visual objects that have XVW_SELECTABLE set to true
XVW_GROUP_DELETE (N/A)	xvobject	N/A	Visual objects that have XVW_SELECTABLE set to true, and a parent currently in *edit mode*.
XVW_GROUP_DELETE_ALL (N/A)	int	FALSE	TRUE/FALSE
XVW_MENUABLE (menuable)	int	FALSE	TRUE/FALSE
XVW_NUM_CHILDREN (N/A)	int	N/A	number of children (values >= 0) of the specified object.
XVW_NUM_SELECTIONS (N/A)	int	N/A	number of selected children (values >= 0)
XVW_REFRESH_CHILDREN_ON_EDIT (refreshChildrenOnEdit)	int	TRUE	TRUE/FALSE
XVW_RESIZABLE (resizable)	int	FALSE	TRUE/FALSE
XVW_SELECTABLE (selectable)	int	TRUE	TRUE/FALSE

Descriptions of Attributes That Enable Direct Manipulation of Children			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_SELECTIONS (N/A)	xvobject *	NULL	array of xvwidgets that are currently selected
XVW_SELECT_ADD (N/A)	xvobject	N/A	Visual objects that have XVW_SELECTABLE set to true, and a parent currently in *edit mode*.
XVW_SELECT_ADD_ALL (N/A)	int	FALSE	TRUE/FALSE
XVW_SELECT_CALLBACK (N/A)	void (*call-back_rou-tine)(xvob-ject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_SELECT_DELETE (N/A)	xvobject	N/A	Visual objects that have XVW_SELECTABLE set to true, and a parent currently in *edit mode*.
XVW_SELECT_DELETE_ALL (N/A)	int	FALSE	TRUE/FALSE
XVW_SELECT_REPLACE (N/A)	xvobject	N/A	Visual objects that have XVW_SELECTABLE set to true, and a parent currently in *edit mode*.
XVW_SNAP_ON (Xsnap)	int	10	values > 0
XVW_UNSELECT_CALLBACK (N/A)	void (*call-back_rou-tine)(xvob-ject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_XSNAP (Xsnap)	int	10	values > 0
XVW_YSNAP (Ysnap)	int	10	values > 0

E.3. Attributes of the VisiQuest 2001 Manager Gadget

As explained at the beginning of the section, the VisiQuest 2001 Manager gadget is not created by the application program. It's only purpose is to act as a superclass for visual objects that are implemented as gadgets, such as the circle object, the line object, the labelstring object, the plot object, the axis object, and so on. Because of its role as a superclass, it offers attributes that subclassed objects can set. These attributes are listed here.

Summary of Attributes Inherited From the Manager Gadget	
Attribute	**Description**
XVW_FORCE_REDISPLAY	

Descriptions of Attributes Inherited From the Manager Gadget			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_FORCE_REDISPLAY (forceRedisplay)	int	FALSE	TRUE/FALSE

F. Callbacks, Event/Action/Input Handlers, & Timeouts

VisiQuest 2001, like the X Window system on which it is based, is *event driven*. Events occur, and applications are expected to respond appropriately. Some events are derived from user input; these events include key press events, mouse click events, and pointer motion events. Other events are derived from interaction between two different applications; the classic example of this is exposure events, which occur when a window of one program is moved and exposes a window from a second program. Still other events may be initiated via the window manager, such as mapping and unmapping events.

There is no way to predict which events will occur, or when they will occur -- that depends on the user. Therefore, it is natural for an application developer to tell the underlying system, "when *something happens*, call *this routine*." To be more realistic, the application is more likely to need to say, "when this *particular thing* happens, call this routine." Since subroutines are usually in need of parameters, it is easy to see where the request would be extended to, "when this particular thing happens, call this routine *with this data"*

The method that VisiQuest 2001 applications use to make such requests of the underlying VisiQuest 2001 libraries is with the installation of *callbacks*, *event handlers*, *action handlers*, *input handlers*, and *timeouts*. These are simply specialized subroutines with predefined parameter lists that are automatically called when the specified event happens on the specified object. The type of object in which the event occurs, as well as the type of event itself, determines whether the routine is classified as callback, an event handler, an action handler, or an input handler.

Callbacks are used with GUI objects. The event which causes the callback to be "fired," or called, is defined by the GUI object itself. Event handlers may be used with GUI objects, but more frequently they are used with visual objects. The event which causes an event handler to be called is defined by the calling application; if desired, an event handler may be called when any one of several events occurs. The events that evoke event handlers are fairly general; they are events such as "the pointer was moved" or "a key was pressed." On the other hand, action handlers support very specific combinations of events, such as "the return key was pressed," or "the shift key was held down while the first mouse button was pressed." In contrast, input handlers are invoked when data changes rather than when an event occurs on the X display. An input handler can called

when a particular file is changed, or when input (or output) is detected on the a descriptor.

F.1. Using Callbacks

GUI objects present the user with information, and passively prompt the user to perform a particular action. The type of action is defined by the nature of the GUI object; a button object, for example, waits for a button press. If and when the user performs the specified action, the GUI object can then pass software control to a subroutine defined by the application. The subroutine to which software control is passed when the specified action is performed on the GUI object by the user is called the *callback*.

To install a callback on a particular GUI object, the *xvw_add_callback()* routine is used:

F.1.1. xvw_add_callback() — *add a callback to a GUI object*

Synopsis

```
void xvw_add_callback(
   xvobject   object,
   char       *type,
   void       (*callback_routine)(xvobject, kaddr, kaddr),
   kaddr      client_data)
```

Input Arguments

`object`
object on which to add callback.

`type`
event type on which to add callback, one of:

```
XVW_BUTTON_SELECT
XVW_DESTROY
XVW_LIST_HIGHLT_ELEM
XVW_LIST_ITEM_ACTION
XVW_LIST_ITEM_SELECT
XVW_LIST_UNHIGHLT_ELEM
XVW_SCROLL_CONT_MOTION
XVW_SCROLL_INCR_MOTION

XVW_DOUBLE_CALLBACK
XVW_EDITMODE_CALLBACK
XVW_ERROR_CALLBACK
XVW_FLOAT_CALLBACK
XVW_GEOMETRY_CALLBACK
XVW_HELP_CALLBACK
XVW_INFO_CALLBACK
XVW_INPUTFILE_CALLBACK
```

```
XVW_INTEGER_CALLBACK
XVW_LAYOUT_CALLBACK
XVW_OUTPUTFILE_CALLBACK
XVW_SELECT_CALLBACK
XVW_TEXTDISPLAY_CALLBACK
XVW_TEXTINPUT_CALLBACK
XVW_WARN_CALLBACK
```

`callback_routine`
callback routine to install

`client_data`
pointer to private application data that will be passed to callback routine

Description

Installs a callback on a GUI object. When the GUI object is used as specified by the *type* argument, the callback will be called.

A callback *must* be defined as follows:

```
void callback(
    xvobject object,
    kaddr  client_data,
    kaddr  call_data)
```

object -
The GUI object for which the callback was invoked.

client_data -
The pointer to the private client data, used to pass parameters from the application to the callback routine.

call_data -
The *call_data* is the mechanism through which the GUI object itself can pass parameters to a callback when * ! applicable. The structure type of the ca
defined by the GUI object; it must be cast to its correct structure type before being used in the callback. Please see the documentation on the particular GUI object of interest for the definition and use of the call_data pointer by a particular GUI object.

To delete a callback currently installed on a GUI object, the *xvw_remove_callback()* routine is used:

F.1.2. xvw_remove_callback() — *remove a callback from a GUI object*

Synopsis

```
void xvw_remove_callback(
```

```
xvobject   object,
char       *type,
void       (*callback_routine)(xvobject, kaddr, kaddr),
kaddr      client_data)
```

Input Arguments

`object`

GUI object from which to remove callback.

`type`

event type from which to remove callback, one of:

```
XVW_BUTTON_SELECT
XVW_DESTROY
XVW_LIST_HIGHLT_ELEM
XVW_LIST_ITEM_ACTION
XVW_LIST_ITEM_SELECT
XVW_LIST_UNHIGHLT_ELEM
XVW_SCROLL_CONT_MOTION
XVW_SCROLL_INCR_MOTION

XVW_DOUBLE_CALLBACK
XVW_EDITMODE_CALLBACK
XVW_ERROR_CALLBACK
XVW_FLOAT_CALLBACK
XVW_GEOMETRY_CALLBACK
XVW_HELP_CALLBACK
XVW_INFO_CALLBACK
XVW_INPUTFILE_CALLBACK
XVW_INTEGER_CALLBACK
XVW_LAYOUT_CALLBACK
XVW_OUTPUTFILE_CALLBACK
XVW_SELECT_CALLBACK
XVW_TEXTDISPLAY_CALLBACK
XVW_TEXTINPUT_CALLBACK
XVW_WARN_CALLBACK
```

`client_data`

pointer to private application data that was being passed to callback routine

Description

Removes a callback from a GUI object.

F.1.3. Callback Example

```
#include "design.h"

/*
 * This example illustrates how a callback is installed on a button object.
 * It also reminds us of the fact that we need not use the client_data
 * pointer to pass any information that can be obtained through a
 * xvw_get_attributes() calls on the xvobject itself.
 *
 */

void button_callback PROTO((xvobject, kaddr, kaddr));

void main(
   int  argc,
   char *argv[])
{
     xvobject   manager;
     xvobject   button;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidgets library */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror("example", "main", "Cannot open display");
           kexit(KEXIT_FAILURE);
     }

     /* create a manager backplane */
     manager = xvw_create_manager(NULL, "back");

     /* a single button on the manager */
     button = xvw_create_button(manager, "button");
     xvw_set_attributes(button,
           XVW_LABEL,            "Yellow", /* label says yellow  */
           XVW_BACKGROUND_COLOR, "yellow", /* yellow color       */
           NULL);

     /* add callback to button */
     xvw_add_callback(button, XVW_BUTTON_SELECT,
                   button_callback, NULL);

     /* display and run */
     xvf_run_form();
}

/*
 * here's the callback for the button
 */
void button_callback(
    xvobject object,      /* the button */
    kaddr    client_data, /* not used   */
    kaddr    call_data)   /* not used   */
{
```

```
        float char_width;
        float char_height;
        char  *label;
        char  *bg_color;

        /*
         *  obtain info from the button object
         */
        xvw_get_attributes(object,
             XVW_CHAR_WIDTH,       &char_width,
             XVW_CHAR_HEIGHT,      &char_height,
             XVW_BACKGROUND_COLOR, &bg_color,
             XVW_LABEL,            &label,
             NULL);

        /*
         *  use kinfo() to pop up information about the button
         */
        kinfo(KSTANDARD, "Button %s,\ncolor %s,\nwidth x height = %.1f x %.1f",
              label, bg_color, char_width, char_height);
    }
```

F.2. Using Event Handlers

Visual objects present the user with information. Unlike GUI objects, they do not necessarily imply a particular action on the part of the user. For example, the button GUI object passively prompts the user to click the button on it. By contrast, the image visual object just sits there, refreshing itself when necessary. Depending on context, it may or may not make sense for the user to click on the image.

When visual objects are used, it is up to the application to decide whether a particular event, such as a button press or a pointer motion, will invoke any particular subroutine. The application specifies not only which visual object will invoke which subroutine (as in a callback) but also for which event(s) the subroutine will be invoked (in the case of event handlers only). The fact that the application specifies which event which will invoke the subroutine makes the subroutine an *event handler*, as opposed to a *callback* in which the object itself would define the event.

To install an event handler for a particular event on a visual object, the *xvw_add_event()* or *xvw_insert_event()* routine is used.

F.2.1. xvw_add_event() — *add an event handler to an object*

Synopsis

```
void xvw_add_event(
   xvobject      object,
   unsigned long event_mask,
   void          (*routine)(xvobject, kaddr, XEvent *, int *),
   kaddr         client_data)
```

Input Arguments

`object`
object on which to install event handler (note that you may not pass NULL)

`event_mask`
event mask representing the event for which the event handler is to be fired

`routine`
the event handler to be called when event occurs

`client_data`
private data to be used by the event handler

Description

Add an event handler to a visual object. When the specified event(s) occur, the event handler will be called.

Since the X Toolkit cannot support event handling of gadgets, we support them by placing the event handler on the parent and then directing the dispatch of the event handler directly.

The event handler *must* be associated with an object; only when the specified event(s) occur on the specified object will the event handler be called (ie, the same event in another object will be ignored).

An event handler must be declared in the following form:

```
void event_handler(
   xvobject object,
   kaddr  client_data,
   XEvent *event,
   int *dispatch)
```

object -
The object for which the event handler was invoked. It will not be NULL.

client_data -
The pointer to the client data, used to pass parameters from the application to the event handler.

event -
This is a pointer to the XEvent union which caused the event handler to be invoked. For details on the XEvent union, see Chapter 8 of the *Xlib Programming Manual*, by Adrian Nye; the definition of the XEvent union is on page 232.

dispatch -
By default, the event that caused this event handler to be invoked will continue to propagate to any other event handlers that might also be installed for the same event on the same visual object. The *dispatch* integer pointer can be used to prevent the event from being

dispatched, and to prevent any other such event handlers from being called. To prevent dispatch of the event to any other event handlers, set this variable to FALSE, as in: `*dispatch = FALSE;`

F.2.2. xvw_insert_event() — *insert an event handler into an object's event list.*

Synopsis

```
void xvw_insert_event(
   xvobject      object,
   unsigned long event_mask,
   void          (*routine)(xvobject, kaddr, XEvent *, int *),
   kaddr         client_data,
   int            position)
```

Input Arguments

`object`
: object on which to install event handler

`event_mask`
: event mask representing the event for which the event handler is to be fired

`routine`
: the event handler to be called when event occurs

`client_data`
: private data to be used by the event handler

`position`
: the position to which this event handler is to be added to the event list. Event handlers for a particular object are maintained in an event list for the object. When multiple event handlers are installed on the same object, they are fired in the order in which they appear in the list. The position may be set to KLIST_HEAD to insert the event at the head of the list, or KLIST_TAIL to insert the event handler at the end of the list. Note: xvw_add_event() adds the event handler at the end of the list.

Description

Adds an event handler to an object, but also allows specification of the position in the event list where the event handler is to be added. See xvw_add_event() for general details.

In fact, *xvw_add_event()* is a macro that simply calls *xvw_insert_event()* with position set to KLIST_TAIL.

Note that both the *xvw_add_event()* and *xvw_insert_event()* routines take an *event mask*. The event mask indicates which event(s) will invoke the event handler. For more information on the event mask, please see Chapter 8 of the *Xlib Programming Manual*, by Adrian Nye. Some of the more common event masks include: `KeyPressMask, KeyReleaseMask, ButtonPressMask, ButtonReleaseMask,`

PointerMotionMask, PointerMotionHintMask, ButtonMotionMask, Button1MotionMask, Button2MotionMask, Button3MotionMask EnterWindowMask, LeaveWindowMask, FocusChangeMask, ExposureMask, VisibilityChangeMask, and StructureNotifyMask.

To delete an event handler that is currently installed on a visual object, the *xvw_remove_event()* routine is used:

F.2.3. xvw_remove_event() — *remove an event handler from an object*

Synopsis

```
void xvw_remove_event(
   xvobject      object,
   unsigned long event_mask,
   void          (*routine)(xvobject, kaddr, XEvent *, int *),
   kaddr         client_data)
```

Input Arguments

`object`
: visual object from which to remove installed event handler

`event_mask`
: event mask representing the event for which the event handler was being fired

`routine`
: the event handler to remove

`client_data`
: private data being used by the event handler

Description

Removes an event handler from a visual object.

The following example creates a red marker visual object. It installs an event handler which will turn the marker from red to blue when the button is pressed on the marker, and will run the marker back to red when the button is released. In this example, it is necessary to pass an array of strings to the event handler; this is done via the client_data argument.

F.2.4. Event Handler Example

```
#include <envision.h>

void marker_event_handler PROTO((xvobject, kaddr, XEvent *, int *));

/*
 * This is an example of a marker object using an event handler.
 * It creates a single marker annotation, and adds an event handler, so that
 * button press on the marker turns it blue!  Button release turns it red again!
 */
```

```
void main(
   int  argc,
   char *argv[])
{
     xvobject marker;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

     /* initialize the xvwidget library */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
           kerror(NULL, "main", "unable to open display");
           kexit(KEXIT_FAILURE);
     }

     /* create the bowtie marker in red at (0.5, 0.5) with scale of 4 */
     marker = xvw_create_marker(NULL, "marker");
     xvw_set_attributes(marker,
          XVW_MARKER_XPLACEMENT,    0.5,
          XVW_MARKER_YPLACEMENT,    0.5,
          XVW_FOREGROUND_COLOR,     "red",
          XVW_GRAPHICS_MARKERTYPE,  KMARKER_BOW_TIE,
          XVW_GRAPHICS_MARKERSCALE, 4,
          NULL);

     /*
      * Add the event handler "marker_event_handler" on the marker,
         * which will be invoked on both button press & button release.
      */
     xvw_add_event(marker, ButtonPressMask | ButtonReleaseMask,
                marker_event_handler, NULL);

     /* display and run */
     xvf_run_form();
}

/*
 *  here is the event handler invoked by button press &
 *  button release events that happen on the marker.
 */
void marker_event_handler(
    xvobject  object,
    kaddr     client_data,
    XEvent    *event,
    int       *dispatch)
{
     if (event->type == ButtonPress)
     {
        xvw_set_attribute(object, XVW_FOREGROUND_COLOR, "blue");
        kfprintf(kstderr, "Turning marker blue\n");
     }
     else if (event->type == ButtonRelease)
     {
        xvw_set_attribute(object, XVW_FOREGROUND_COLOR, "red");
        kfprintf(kstderr, "Turning marker red\n");
     }
}
```

F.3. Using Action Handlers

Action handlers are similar to event handlers; the difference is that the action that is specified for an action handler is more specific than the event that is specified for an event handler. For example, an event handler may be used to specify that control flow be diverted to a subroutine when *any* key on the keyboard is pressed, by installing the event handler with the event mask "KeyPress". However, this may not be specific enough; in some cases, it is necessary to indicate *which* key press which will divert control to the handler. In this case, an action handler might be installed on the object, where the action could be specified as the carriage return, a capital "Q", or the "Escape" key. Compound actions can also be specified, such as the first mouse button pressed in conjunction with the key "Y" being pressed, or the third mouse button released in conjunction with the META key being pressed.

To install an action handler for a particular event on a GUI or visual object, the following routines are used.

F.3.1. xvw_add_action() — *add an action handler to an object*

Synopsis

```
void xvw_add_action(
   xvobject   object,
   char       *action,
   void      (*routine)(xvobject, kaddr, XEvent *),
   kaddr      client_data,
   int        override)
```

Input Arguments

`object`
: object on which to add action handler (note that you may not pass NULL)

`action`
: the action which will invoke the action handler

`routine`
: the action handler to install

`client_data`
: private data to be used by action handler

`override`
: TRUE if the action specified should override any previously installed action handlers, FALSE if the action specified should augment any previously installed action handlers

Description

Adds an action handler to a GUI or visual object. When the specified action occurs, the action handler will be called.

Since the X Toolkit cannot support action handling on gadgets, we support them by placing the action handler on the parent of the gadget and then directing the dispatch of the action handler directly.

The action handler *must* be associated with an object; only when the specified action(s) occur in the specified object will the action handler be called (ie, the same action in another object will be ignored).

An action handler must be declared in the following form:

```
void action_handler(
   xvobject object,
   kaddr  client_data,
   XEvent *event)
```

object -
The object for which the action handler was invoked. It will not be NULL.

client_data -
The pointer to the client data, used to pass parameters from the application to the action handler.

event -
This is a pointer to the XEvent union which caused the action handler to be invoked. For details on the XEvent union, see Chapter 8 of the *Xlib Programming Manual*, by Adrian Nye; the definition of the XEvent union is on page 232.

Note that the *xvw_add_action()* routine takes an *action string*. The action string indicates which action will invoke the action handler. The action string contains triangular brackets "<" and ">" around the desired action; it may also contain optional modifiers, and in the case of some action, by a "detail" field that specifies additional information about the action.

For those who are familiar with X Toolkit programming, the action string is simply the first part of a translation table which specifies the event. For more information on action strings, please see Chapter 7.1 of the *X Toolkit Intrinsics Programming Manual*, by Adrian Nye and Tim O'Reilly.

Examples of some common action strings include:

<Btn1Up>(2) Invoke the handler on double-click of Button1.
Shift<Btn2Down> Invoke the handler on Button2 click if the shift key is down.
<Key>Return Invoke the action handler when the carriage return key is pressed
<Meta>Q Invoke the handler when the Q key is pressed together with the Meta key
<Ctrl>C Invoke the handler on CTRL-C
<Key>? Invoke the action handler when the question mark is pressed.

Button Press Actions

These actions specify a mouse button press while focus is in the object:
<BtnDown>, <Btn1Down>, <Btn2Down>, <Btn3Down>

Button Release Actions

These actions specify a mouse button release while focus is in the object:
<BtnUp>, <Btn1Up>, <Btn2Up>, <Btn3Up>

Key Press Actions:

These actions specify a key press while focus is in the object:
<Key>, <KeyDown>, <Ctrl>, <Meta>, <Shift>
Note that <Key> refers to *any* key being pressed, while <Key>x refers to the "x" key being pressed.

KeyRelease Actions

This action specifies a key release while focus is in the object:
<KeyUp>
Note that <KeyUp> refers to *any* key being released, while <Key>x refers to the "x" key being released.

MotionNotify Actions:

These actions specify movement of the pointer in the object:
<Motion>, <BtnMotion>, <Btn1Motion>, <Bt21Motion>, <Btn3Motion>

EnterNotify:

This action specifies movement of the pointer into the object:
<Enter>

LeaveNotify:

This action specifies movement of the pointer out of the object:
<Leave>

The actions listed above are the most common; there are others, which may be found in the "Abbreviations" column of Table 7-1 on page 192 of the *X Toolkit Intrinsics Programming Manual*, by Adrian Nye and Tim O'Reilly.

To delete an action handler that is currently installed on a visual object, the *xvw_remove_action()* routine is used:

F.3.2. xvw_remove_action() — *remove an action handler from an object*

Synopsis

```
void xvw_remove_action(
   xvobject   object,
   char       *action,
   void      (*routine)(xvobject, kaddr, XEvent *),
   kaddr      client_data)
```

Input Arguments

`object`
: object from which to remove action handler

`action`
: the action which was invoking the action handler

`routine`
: the action handler to de-install

`client_data`
: pointer to private application data that was being passed to action handler

Description

Removes an action handler from a GUI or visual object.

F.3.3. Action Handler Example

This example demonstrates how an action handler can be installed on a hybrid xvroutine. Here, the action handler is used to quit the program when the user hits 'q', since a hybrid *xvroutine* has no formal GUI.

F.4. Using Input Handlers

Sometimes it is necessary for an application to perform a particular action when a certain file has been updated externally. The *xvwidgets* library supports this capability via a file detection mechanism. If and when the file in question is updated, software control can be passed to a subroutine defined by the application. The subroutine to which software control is passed when the file being monitored undergoes a change is called an *input handler*.

The *xvwidgets* library allows an input handler to be installed on a file where the file is specified either by its name or by its file identification number. When the input handler is to be installed on the file using the filename, the following routine is used:

F.4.1. xvw_add_detectfile() — *add a (file) detect handler to an object*

Synopsis

```
void xvw_add_detectfile(
   xvobject  object,
   char      *filename,
   double    argtime,
   int       (*routine)(xvobject, char *, kaddr),
   kaddr     client_data)
```

Input Arguments

`object`
object on which to place the detect handler. Pass NULL if the detect handler is to be invoked in general, and not associated with any particular object.

`filename`
the name of the file to be monitored.

`argtime`
the interval (in seconds) at which the file should be checked for changes.

`routine`
the detect handler routine to be called when change to the file is detected.

`client_data`
pointer to client data that will be passed to detect handler

Description

Causes a detection mechanism to be installed on the specified file. After the specified interval of time has elapsed, the file will be checked for any modification. If a modification to the file is detected, then the specified detect handler is called.

The detect handler can be associated with an object, so that file detection is automatically discontinued when the object is destroyed. If NULL is passed for the object, then the detect handler is added to the global file detection list.

The detect file callback must be declared in the following form:

```
int detect_handler(
     xvobject object,
     char *filename,
     kaddr client_data)
```

object -
If *xvw_add_detectfile()* is called with a particular xvobject, that object will be passed into the detect handler.

filename -
This is the name of the file being monitored for change.

client_data -
The pointer to the client data, used to pass parameters from the application to the detect handler.

To discontinue monitoring of a file currently being monitored, and to remove an input handler currently installed a file using its filename, the following routine is used:

F.4.2. xvw_remove_detectfile() — *remove a (file) detect handler from an object*

Synopsis

```
void xvw_remove_detectfile(
   xvobject  object,
   char      *filename,
   int       (*routine)(xvobject, char *, kaddr),
   kaddr     client_data)
```

Input Arguments

`object`
object in which to remove the detect file handler, if NULL then removed from the global list.

`filename`
the name of the file that was being monitored.

`routine`
the detect handler routine which was being called when change to the file was detected.

`client_data`
pointer to private application data that was being passed to action handler

Description

Causes the detection mechanism previously installed with *xvw_add_detectfile()* to be removed from the specified file descriptor.

If the detect handler was associated with an object, the file detection will be automatically removed when the object is destroyed. Alternatively, it can be removed before the object is destroyed by using this routine. If NULL was passed to *xvw_add_detectfile()* for the object, then you must call this routine if you need to remove the installed detect handler.

As an alternative to monitoring a file that is specified by its filename, an input handler may be installed on a file where the file is specified by its file identification number. Note that both stdin and stdout can be monitored in this way. When the input handler is to be installed on the file using this method, the following routine is used:

F.4.3. xvw_add_detectfid() — *add (fid) input handler to an object*

Synopsis

```
void xvw_add_detectfid(
   xvobject   object,
   int        fid,
   void       (*routine)(xvobject, int, kaddr),
   kaddr      client_data)
```

Input Arguments

`object`
: object on which to place the input handler. Pass NULL if the input handler is to be invoked in general, and not associated with any particular object.

`fid`
: the file descriptor which will be monitored for input (or output)

`routine`
: the input handler routine to be called when change on the fid is detected

`client_data`
: pointer to client data that will be passed to input handler

Description

Causes a detection mechanism to be installed on the specified file descriptor; the file descriptor will be polled intermittently for change (input or output). If a change in the file descriptor is detected, then the specified input handler is called.

The input handler can be associated with an object, so that fid detection is automatically discontinued when the object is destroyed. If NULL is passed for the object, then the input handler is added to the global file detection list.

The detect file callback must be declared in the following form:

```
int input_handler(
     xvobject object,
     int      fid,
     kaddr    client_data)
```

object -
If *xvw_add_detectfid()* is called with a particular xvobject, that object will be passed into the input handle

r.

fid -
This is the file descriptor being monitored for change.

client_data -
The pointer to the client data, used to pass parameters from the application to the input handler.

To discontinue monitoring of a file descriptor currently being monitored, and to remove an input handler cur-

rently installed, the following routine is used:

F.4.4. xvw_remove_detectfid() — *remove (fid) input handler from an object*

Synopsis

```
void xvw_remove_detectfid(
   xvobject   object,
   int        fid,
   void       (*routine)(xvobject, int, kaddr),
   kaddr      client_data)
```

Input Arguments

`object`
: object for which to remove the input file handler, or NULL if the input handler was being invoked in general, not associated with a particular object.

`fid`
: the file descriptor on which input (or output) was being detected.

`routine`
: the input handler that was being called when change on the fid was detected

`client_data`
: pointer to client data which was being passed to input handler

Description

Causes the detection mechanism previously installed with *xvw_add_detectfid()* to be removed from the specified file descriptor.

If the input handler was associated with an object, the file detection will be automatically removed when the object is destroyed. Alternatively, it can be removed before the object is destroyed by using this routine. If NULL was passed to *xvw_add_detectfid()* for the object, then you must call this routine if you need to remove the installed input handler.

F.5. Using Timeouts

Timeouts provide a mechanism to redirect software control flow to a particular routine after a specified time interval. Depending on how the value of the *stop_timer* argument is set within the timeout, the timeout may be called once, a number of times until a particular condition is met, or may be repeated indefinitely.

Sometimes a timeout is associated with a particular object. For example, the image object uses a timeout to check the file containing the displayed image every so often, so that it can update the displayed image if the contents of the file change.

In other instances, a timeout is not associated with a particular object, but simply gets called after a particular amount of time passes, regardless of the objects that are displayed.

To install a timer for a particular time interval, the *xvw_add_timeout()* routine is used.

F.5.1. xvw_add_timeout() — *add a timeout to an object*

Synopsis

```
void xvw_add_timeout(
   xvobject   object,
   double     interval,
   void      (*routine)(xvobject, kaddr, int *),
   kaddr      client_data)
```

Input Arguments

`object`
: object on which to install timeout

`interval`
: the time interval (in seconds) after which the timeout will be called.

`routine`
: the timeout routine to be installed

`client_data`
: private data to be used by the timeout

Description

Adds a timeout to a GUI or visual object. After the specified interval of time has elapsed, the timeout will be called. Since the X Toolkit cannot support timeouts on gadgets, we support them by placing the timeout on the parent and then directing the dispatch of the timeout directly.

The timeout can be associated with an object, so that it will be automatically removed when the object is destroyed. If NULL is passed for the object, then the timeout is added to the global file detection list.

An timeout must be declared in the following form:

```
void timeout(
   xvobject object,
   kaddr    client_data,
   int      *stop_timer)
```

object -
If *xvw_add_timeout()* is called with a particular xvobject, that object will be passed into the timeout.

client_data -

The pointer to the client data, used to pass parameters from the application to the timeout.

stop_timer -
By default, the timeout will be invoked again after the specified time interval passes once more. To stop the timeout from being called after the next time interval is up, set the stop_timer to TRUE, as in:
`*stop_timer = TRUE;`

Note that the *xvw_add_timeout()* routine takes an *interval*. The interval indicates how often, in seconds, the timeout is to be called. If the interval was set to 0.2, this would indicate that the timeout was to be called every 0.2 seconds, or 5 times a second.

To remove a currently installed timeout, the *xvw_remove_timeout()* routine is used:

F.5.2. xvw_remove_timeout() — *removes a timeout from an object*

Synopsis

```
void xvw_remove_timeout(
   xvobject   object,
   void     (*routine)(xvobject, kaddr, int *),
   kaddr      client_data)
```

Input Arguments

`object`
object from which to remove installed timeout
`routine`
the timeout to remove
`client_data`
private data being be used by the timeout

Description

Remove a timeout from a object or gadget.

F.5.3. Timeout Example

The following example uses a timeout:

```
#include "design.h"

/*
 * In this example, a timeout is used to update a label object every second.
 */
```

```
void timeout PROTO((xvobject, kaddr, int *));

void main(
   int  argc,
   char *argv[])
{
     xvobject   manager;
     xvobject   label;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidgets library */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror("example", "main", "Cannot open display");
           kexit(KEXIT_FAILURE);
     }

     /* create a manager backplane */
     manager = xvw_create_manager(NULL, "back");

     /* a single button on the manager */
     label = xvw_create_label(manager, "label");
     xvw_set_attributes(label,
                  XVW_LABEL,  "0 seconds have passed",
                  XVW_FORCE_REDISPLAY, TRUE,
                  NULL);

     /* add callback to button */
     xvw_add_timeout(label, 1.0, timeout, NULL);

     /* display and run */
     xvf_run_form();
}

/*
 * here's the timeout
 */
void timeout(
    xvobject object,
    kaddr    client_data,
    int      *stop_timer)
{
     char   temp[KLENGTH];
     static int count = 1;

     ksprintf(temp, "%d seconds have passed", count++);
     xvw_set_attribute(object, XVW_LABEL, temp);
}
```

F.6. About Client Data

Callbacks, event handlers, action handlers, input handlers, and timeouts all pass parameters via a *client data pointer*. The client data pointer serves as a mechanism with which the application can bundle up all the

parameters and pass them to the appropriate installation function (*xvw_add_callback()*, *xvw_add_event()*, *xvw_add_action()*, *xvw_add_detectfile()*, *xvw_detectfid()*, or *xvw_add_timeout()*, depending on context). Internal routines in the *xvwidgets* library are responsible for passing the client data along to the installed handler when it is called. Inside the handler, the client data pointer is cast back to the appropriate data type before it is used.

Only a pointer may be used as client data; it is defined as type *kaddr*, which is the VisiQuest 2001 method of specifying a pointer of unspecified data type. The type of pointer is defined by the application; it might be a pointer to a scalar, a pointer to an array, or a pointer to a data structure that is defined by the application.

First, decide on the number and data type of the arguments that will be needed by the callback, event handler, action handler, or input handler.

If the handler needs NO arguments, pass NULL as the client data parameter to the installation function. Inside the handler, the client data argument will remain unused.

If the handler needs a single argument of an already-defined data type, declare a variable which is a pointer to the desired data type in the installation function. Allocate the pointer if necessary, and pass the initialized pointer as the client data parameter to the installation function. Inside the handler, cast the client data argument to the pointer type, and use the pointer variable as desired.

If the handler needs more than one argument, it will be necessary to define a new data structure, one that contains an element of appropriate data type for each of the required arguments. Be sure that both the routine which calls the installation function and the handler itself has access to the new data structure. Declare a variable which is a pointer to the data structure in the installation function. Dynamically allocate the pointer, and initialize each field with the value that you would have used if you had been preparing to pass the parameters to the handler "normally". Pass the pointer to the allocated, initialized data structure as the client data to the installation function.

Then, inside the handler, cast the client data pointer to the same data type that you used when allocating and initializing it. After the client data pointer has been cast, the contents can be referenced normally and the values will be the same as when they were initialized. anything as the client data.

F.6.1. Client Data Example 1

In this example, an event handler is installed on an image visual object. The event handler will quit the program when the user does a button press. Here, we need to pass the image data stored in the *kobject* structure as the client data; however, kobjects can be passed to event handlers directly (without using "&") since they are, by definition, already pointers.

```
#include <design.h>

/*
 * This program demonstrates how to use an event handler and
 * use the client_data pointer to pass in a parameter.
 * It creates a colored backplane and adds an event handler
 * that allows the user to quit the program by clicking the mouse
 * in the backplane; client_data is used to pass in the color name
 */

void  quit_program PROTO((xvobject, kaddr, XEvent *, int *));
```

```
void main(
   int  argc,
   char *argv[])
{
        xvobject backplane; /* the manager object */
     char     *color = "navy";

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidgets library */
        if (!xvw_initialize(XVW_MENUS_XVFORMS))
        {
           kerror(NULL, "main", "unable to open display");
           kexit(KEXIT_FAILURE);
        }

     /* create the manager object (default toplevel) */
        backplane = xvw_create_manager(NULL, "backplane");

     /* specify size, make the backplane coral */
        xvw_set_attributes(backplane,
                  XVW_WIDTH,  300,
                  XVW_HEIGHT, 300,
                  XVW_BACKGROUND_COLOR, color,
                  NULL);

        /*
         * add the event handler to quit the program. pass in the
      * color string as the client_data so we can use it inside
      * the event handler.
         */
        xvw_add_event(backplane, ButtonPressMask, quit_program, color);

     /* display and run */
        xvf_run_form();
}

/*
 *  Event handler to quit program on Button Press.
 *  Uses the client_data pointer so that it has access
 *  to the color string.
 */
void  quit_program(
   xvobject object,       /* the backplane */
   kaddr    client_data,  /* used to pass in the color */
   XEvent   *event,       /* also not used - we know what the event was. */
   int      *dispatch)    /* if we didn't want any other event handler to
                  be called after this one is finished, we'd set
                  *dispatch = FALSE */
{
     char *color = (char *) client_data;

     kinfo (KSTANDARD, "This was a %s backplane.", color);
        xvw_destroy(object);
        kexit(KEXIT_SUCCESS);
}
```

F.6.2. Client Data Example 2

In this example, the same callback is installed on a set of button GUI objects; the callback uses the integer button ID number (passed in as client_data) in the information that is popped up.

```
#include "design.h"


/*
 *   This example program puts up 5 buttons in a diagonal pattern.
 *   It adds very simple callbacks to the buttons. The callback uses
 *   the integer button ID number (passed in as client_data) in the
 *   information that is popped up.
 */
static void button_cb PROTO((xvobject, kaddr, kaddr));

#define BUTTON_NUM 5

void main(
   int  argc,
   char *argv[])
{
     int         i;
     int         *button_id;
     xvobject    manager;
     xvobject    button;
     xvobject    offset;
     static char *names[] = {"AAA", "BBB", "CCC", "DDD", "EEE"};

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidget library */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "Cannot open display");
           kexit(KEXIT_FAILURE);
     }

     /* create the manager backplane */
     manager = xvw_create_manager(NULL, "back");

     /* offset will be used to position set of buttons */
     offset = NULL;

     /* create set of buttons */
     for (i = 0; i < BUTTON_NUM; i++)
     {
        /* create the button, set attributes */
        button = xvw_create_button(manager, "button");
        xvw_set_attributes(button,
                XVW_LABEL,        names[i],      /* set label         */
                XVW_RIGHT_OF,     offset,        /* R of last button  */
                XVW_BELOW,        offset,        /* Below last button */
                XVW_CHAR_WIDTH,   10.0,          /* 10 chars wide     */
                XVW_CHAR_HEIGHT,  2.0,           /* 2 char high       */
                XVW_BORDER_WIDTH, 1,             /* thin border       */
                NULL);
```

```
        /* allocate the client data - here, we'll need to pass an int */
        button_id = (int *) kmalloc(sizeof(int));

        /* initialize the integer value */
        *button_id = i+100;

        /* add the callback, passing the integer pointer as client data */
        xvw_add_callback(button, XVW_BUTTON_SELECT, button_cb, button_id);

        /* update offset, to achieve diagonal layout effect */
        offset = button;
     }

     /* display and run */
     xvf_run_form();
}


/*
 *  this is the button callback, which expects an integer pointer
 *  as its client_data so it can print out the button ID number.
 */
static void button_cb (
    xvobject object,
    kaddr    client_data,
    kaddr    call_data)
{
     char *label;
     int  *button_id;  /* client data will be an integer pointer */

     button_id = (int *) client_data; /* cast client_data to int pointer */

     /* to add interest to popup info message, get the button label */
     xvw_get_attribute(object, XVW_LABEL, &label);

     /* use the int pointer as desired, after cast */
     kinfo(KSTANDARD, "Button click on %s, ID number %d\n",
           label, *button_id);
}
```

F.6.3. Client Data Example 3

In this example, the callback installed on a list item selection needs 4 parameters which are passed using the client_data structure.

```
#include "design.h"

/*
 *  In this example, a list GUI object offering a choice of three animals is
 *  displayed.  The user is allowed to click on an item, causing a popup
 *  with information about the animal to be displayed.
 *  The callback installed on list item selection needs an array
 *  of structures passed in, which is allocated and passed into the callback
 *  using the client_data structure.  Because it is a list GUI object, the
 *  call_data pointer is also used.
 */
```

```
/*
 *  here is the structure containing the information we need,
 *  an array of which to be passed to the callback as the client_data
 */
typedef struct _AnimalInfo {

    int   id;
    char  *habitat;
    char  *diet;
    int   endangered;

} AnimalInfo;

/*
 *  defines to identify animal
 */
#define MONKEY   0
#define WOLF     1
#define ELEPHANT 2

/*
 *  prototype for callback
 */
void print_info PROTO((xvobject, kaddr, kaddr));

/*
 *  here's the main program
 */
void main(
   int  argc,
   char **argv,
   char **envp)
{
     xvobject    comp_list;
        xvobject    actual_list;
     xvobject    manager;
     int         foreign;
     AnimalInfo **animal_info;

     static char *animals[] = { "monkey", "wolf", "elephant" };
     int         num = knumber(animals);

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror("example", "main", "Cannot open display");
           kexit(KEXIT_FAILURE);
     }

     /* create manager backplane, width 10, height 4 */
     manager = xvw_create_manager(NULL, "back");
     xvw_set_attributes(manager,
                  XVW_CHAR_WIDTH, 15.0,
                  XVW_CHAR_HEIGHT, 4.0,
                  NULL);
```

```
    /* create compound list object, tack it to parent */
    comp_list = xvw_create_list(manager, "comp_list");
    xvw_set_attribute(comp_list, XVW_TACK_EDGE, KMANAGER_TACK_ALL);

    /* get actual list object from compound list object */
    actual_list = xvw_retrieve_list(comp_list);

    /* add list contents to actual list */
    xvw_change_list(actual_list, animals, num, TRUE);

    /*
     * allocate & initialize client_data
     */
    animal_info = (AnimalInfo **) kcalloc(3, sizeof(AnimalInfo *));

    animal_info[MONKEY] = (AnimalInfo *) kcalloc(1, sizeof(AnimalInfo));
    animal_info[MONKEY]->id         = 1230;
    animal_info[MONKEY]->habitat    = kstrdup("South America");
    animal_info[MONKEY]->diet       = kstrdup("Omniverous");
    animal_info[MONKEY]->endangered = FALSE;

    animal_info[WOLF] = (AnimalInfo *) kcalloc(1, sizeof(AnimalInfo));
    animal_info[WOLF]->id           = 1894;
    animal_info[WOLF]->habitat      = kstrdup("North America");
    animal_info[WOLF]->diet         = kstrdup("Carnivorous");
    animal_info[WOLF]->endangered   = TRUE;

    animal_info[ELEPHANT] = (AnimalInfo *) kcalloc(1,sizeof(AnimalInfo));
    animal_info[ELEPHANT]->id         = 2113;
    animal_info[ELEPHANT]->habitat    = kstrdup("Africa");
    animal_info[ELEPHANT]->diet       = kstrdup("Herbivorous");
    animal_info[ELEPHANT]->endangered = TRUE;

    /*
        * add callback to list selection that will print info.
        * pass the "animal_info" data structure as client_data
        */
    xvw_add_callback(actual_list, XVW_LIST_ITEM_SELECT, print_info,
                     animal_info);

    /* display and run */
    xvf_run_form();
}

/*
 *  callback prints information about the animal chosen from the list
 */
void print_info(
    xvobject button,
    kaddr    client_data,
    kaddr    call_data)
{
    AnimalInfo    **animal_info; /* array of animal info structures */
    xvw_list_struct *list_return; /* holds info about selected item */
    int             animal_index; /* index of selected animal       */
    char            *animal_name; /* name of selected animal        */

    /* cast client data pointer to expected data type (defined by us) */
    animal_info = (AnimalInfo **) client_data;
```

```
        /* cast call_data pointer to data type (defined by list gui object) */
        list_return = (xvw_list_struct *) call_data;

        if ( list_return == NULL || animal_info == NULL ) return;

        /* index and string associated w/ selection stored in list struct */
        animal_index = list_return->list_index;
        animal_name  = list_return->string;

        if ( animal_index == -1 || animal_name == NULL ) return;

        kinfo(KSTANDARD, "Selected %s,\n Index %d,\n ID %d,\n Habitat %s,\n %s Diet",
                 animal_name, animal_index,
                 animal_info[animal_index]->id,
                 animal_info[animal_index]->habitat,
                 animal_info[animal_index]->diet);

        if (animal_info[animal_index]->endangered)
           kinfo(KSTANDARD, "Warning! This is an endangered species");
    }
```

G. General Utilities For Visual & GUI Objects

There are a variety of basic functions that are offered by the *xvwidgets* library for use with visual and GUI objects. Some functions are front-ends for X Toolkit functions; these must always be used instead of their X Toolkit counterparts in order to ensure correctness of an application which depends on the *xvwidgets* library, and in order to maintain its support of all three widget sets. A summary of these functions is given in this section. Other functions are simply provided for convenience. Both types of functions are grouped together in this section.

G.1. xvw_appcontext() — *return the application context associated with a object*

Synopsis

```
XtAppContext xvw_appcontext(
   xvobject object)
```

Input Arguments

`object`

object for which to get the application context

Returns

The application context associated with the object on success, NULL on failure

Description

Returns the application context associated with the object. Note that this is the GUI & Visualization

services equivalent of *XtWidgetToApplicationContext()*. If object is NULL, returns the default application context.

Restrictions

Restrictions on data or input as applicable

G.2. xvw_busy() — *set an object to be busy or not busy*

Synopsis

```
void xvw_busy(
   xvobject object,
   int      busy)
```

Input Arguments

`object`
: the object to make "busy" or "not busy"

`busy`
: TRUE to set object "busy", FALSE to set object "not busy"

Description

Sets an object to be "busy" or "not busy". If an object is busy, then all input events to the object are ignored and a "watch" cursor is installed to alert the user that the object will not accept input.

G.3. xvw_check_managed() — *see if an object is managed*

Synopsis

```
int xvw_check_managed(
   xvobject object)
```

Input Arguments

`object`
: the object to check

Returns

TRUE if the object is being managed, FALSE otherwise

Description

Indicates whether or not an object is currently having its geometry managed by its parent. Note that an object cannot be made visible until it is managed.

G.4. xvw_check_mapped() — *see if an object is mapped*

Synopsis

```
int xvw_check_mapped(
   xvobject object)
```

Input Arguments

`object`

the object to check

Returns

TRUE if the object is mapped, FALSE otherwise

Description

Indicates whether or not an object is currently mapped.

G.5. xvw_check_menuactive() — *see if an object's internal menuform is displayed*

Synopsis

```
int xvw_check_menuactive(
   xvobject object)
```

Input Arguments

`object`

the GUI or visual object to check

Returns

TRUE if the internal menuform for the object is currently displayed, FALSE otherwise

Description

Indicates whether or not the internal menuform for a GUI or visual object is currently being displayed.

G.6. xvw_check_menuexist() — *check if an object has an internal menuform*

Synopsis

```
int xvw_check_menuexist(
   xvobject object)
```

Input Arguments

`object`
the object for which to check for internal menuforms

Returns

TRUE if the object has an internal menuform associated with it, FALSE otherwise.

Description

Checks whether or not a GUI or visual object has an internal menuform associated with it.

G.7. xvw_check_realized() — *see if an object is realized*

Synopsis

```
int xvw_check_realized(
   xvobject object)
```

Input Arguments

`object`
the object to be checked

Returns

TRUE if the object has been realized, FALSE otherwise

Description

Indicates whether or not an object has been realized. When an object is *realized*, it has its windows created on the display, and its final initializations done. The realization of a GUI or visual object is delayed until *xvf_run_form()* is called (if the object is created before the call to *xvf_run_form()*), or until software control flow is diverted back to the *xvf_run_form()* loop (f the object is created after the call to *xvf_run_form()*).

G.8. xvw_check_sensitive() — *see if an object is sensitive*

Synopsis

```
int xvw_check_sensitive(
   xvobject object)
```

Input Arguments

`object`
the object to check

Returns

TRUE if the object is sensitive, FALSE otherwise.

Description

Indicates whether or not an object is currently sensitive. Note that a GUI or visual object may be made insensitive using *xvw_sensitive()*. When an object is made insensitive, it will ignore all attempts at user input; it will also appear "dim" or "stippled" to indicate its insensitive state (a current limitation restricts this capability to widgets only; insensitive gadgets will not accept input, but they cannot provide a visual cue to reflect their insensitive state). All objects are normally sensitive.

G.9. xvw_check_subclass() — *check the subclass of an object*

Synopsis

```
int xvw_check_subclass(
   xvobject object,
   xvclass  wclass)
```

Input Arguments

`object`
object for which to check the subclass class - the class to be checked for

Returns

TRUE if the object has the specified subclass, FALSE otherwise

Description

Checks if the object has the specified subclass. Example1: if the object passed in was a zoom object, and the class given was ImageClass, the routine would return TRUE since the zoom object is subclassed off the image object. Example2: if the object passed in was a zoom object, and the class given was ColorClass, the routine would return TRUE, since the zoom object is subclassed off the image

object and the image object is subclassed off the color object. Example3: if the object passed in was a zoom object, and the class given was PanIconClass, the routine would return FALSE, since the PanIconClass does not appear in the inheritance tree of the zoom object.

G.10. xvw_check_toplevel() — *see if object specified is a toplevel, or see if a toplevel exists*

Synopsis

```
int xvw_check_toplevel(
   xvobject object)
```

Input Arguments

`object`
the object to check, or NULL to check for any toplevels

Returns

TRUE if the object specified is a toplevel object (or if any toplevel objects exist), FALSE otherwise.

Description

If the object is specified as non-NULL, returns whether or not the object is a toplevel object. If NULL is passed for the object parameter, returns whether or not any toplevel objects exist.

G.11. xvw_check_visible() — *see if an object is visible*

Synopsis

```
int xvw_check_visible(
   xvobject object)
```

Input Arguments

`object`
the object to check

Returns

TRUE if the object is visible, FALSE otherwise

Description

Indicates whether or not an object is currently visible.

G.12. xvw_children() — *get the children of an object*

Synopsis

```
xvobject *xvw_children(
   xvobject object,
   xvclass  wclass,
   size_t  *num_children)
```

Input Arguments

`object`
: object for which to find children

`wclass`
: the desired class of the children, or NULL for all children

Output Arguments

`num_children`
: the number of children being returned in the xvobject array

Returns

An array of children upon success, NULL on failure

Description

Returns an array of xvobjects containing the children of the object specified. If desired, an object class may be specified so that only children of the specified class are returned.

Side Effects

The array in which the children returned is allocated, and must be freed by the caller.

G.13. xvw_colormap() — *get the colormap associated with a object*

Synopsis

```
Colormap xvw_colormap(
   xvobject object)
```

Input Arguments

`object`
: object to for which get the colormap; pass NULL to get the default colormap.

Returns

The colormap associated with the object (or the default colormap) on success, NULL on failure

Description

Returns the colormap associated with the object. If the object is passed as NULL, it will return the colormap associated with the default display.

G.14. xvw_class() — *get the class of the object*

Synopsis

```
xvclass xvw_class(
    xvobject object)
```

Input Arguments

`object`

object for which return the class

Returns

The class associated with the object on success, NULL on failure

Description

Returns the class associated with the object.

G.15. xvw_create() — *create a new object*

Synopsis

```
xvobject xvw_create(
    xvobject    parent,
    int          transient_parent,
    int          managed,
    kstring     name,
    xvclass     wclass)
```

Input Arguments

`parent`

the parent object; pass NULL if a default toplevel object is to be created automatically to contain the object.

`transient_parent`
only used if the parent object passed as NULL, TRUE will cause the default toplevel to be created as a transient shell while FALSE will cause the default toplevel to be created as an application shell.

`managed`
TRUE if the object should be managed upon creation, FALSE if the object should only be instaniated

`name`
the object name class - the class of the object

Returns

The newly created GUI or visual object on success, NULL on failure

Description

Creates and returns a new GUI or visual object of the desired object class. Note that use of this generalized routine is not as encouraged as use of the appropriate specific object creation routine. For example, if you would like to create a button object, it is generally easier to use *xvw_create_button()*.

G.16. xvw_destroy() — *destroy an object*

Synopsis

```
void xvw_destroy(
   xvobject object)
```

Input Arguments

`object`
the object to be destroyed

Description

Destroy the visual or GUI object specified. Not only does this destroy the windows used by the object, it also frees all memory associated with the object. You may never refer to the object again, after making this call. Any children of the object will also be recursively destroyed. The parent of the object, however, will be left intact.

G.17. xvw_display() — *returns the display associated with a object*

Synopsis

```
Display *xvw_display(
   xvobject object)
```

Input Arguments

`object`

object for which to identify the display

Returns

The display associated with the object or gadget on success, NULL on failure

Description

Returns the display associated with the object. Note that this is the GUI & Visualization services equivalent of *XtDisplay()*. If object is NULL, returns the default display.

G.18. xvw_duplicate() — *duplicate an object*

Synopsis

```
xvobject xvw_duplicate(
   xvobject    object)
```

Input Arguments

`object`

the object to be duplicated

Returns

The duplicated object on success, NULL on failure.

Description

Duplicates the specified GUI or visual object.

G.19. xvw_font() — *return the font being used by a object*

Synopsis

```
XFontStruct *xvw_font(
   xvobject object)
```

Input Arguments

`object`

object to get the font struct from

Returns

The font struct associated with the object on success, NULL on failure

Description

Returns the display font being used by an object; in other words, this is the setting of the XVW_FONT attribute of the object. If the object specified does not have the XVW_FONTE attribute set, the font being used by its parent will be returned. Recursion up the inheritance tree will be done until a setting for the font is found.

G.20. xvw_fontname() — *return the font name being used by an object*

Synopsis

```
char *xvw_fontname(
   xvobject object)
```

Input Arguments

`object`

object for which to query font name

Returns

The fontname associated with the object on success, NULL on failure

Description

Returns the name of the font being used by an object; in other words, this is the setting of the XVW_FONTNAME attribute of the object. If the object specified does not have the XVW_FONT-NAME attribute set, the font being used by its parent will be returned. Recursion up the inheritance tree will be done until a setting for the font name is found.

G.21. xvw_geometry() — *get the geometry of an object*

Synopsis

```
int xvw_geometry(
   xvobject object,
   int       *x,
   int       *y,
   unsigned int    *width,
   unsigned int    *height,
   unsigned int    *border_width)
```

Input Arguments

`object`
object for which to return geometry

Output Arguments

`x`
x position of the object in pixels, with respect to its parent

`y`
y position of the object in pixels with respect to its parent

`width`
width of the object in pixels

`height`
height of the object in pixels

`border_width`
border width of the object in pixels

Returns

TRUE (1) on success, FALSE (0) on failure

Description

Returns the current geometry of the specified object, including (x,y) position with respect to the parent, width, height, and border width. All geometry values are given in *pixels*.

G.22. xvw_lower() — *lower an object*

Synopsis

```
void xvw_lower(
   xvobject object)
```

Input Arguments

`object`
the object to be lowered

Description

Lowers a GUI or visual object with respect to other siblings in the visual stacking order. If the object overlaps another sibling object, this call will cause the object to be obscured by the sibling.

G.23. xvw_manage() — *manage an object*

Synopsis

```
void xvw_manage(
   xvobject object)
```

Input Arguments

`object`
the object to be managed

Description

Causes the specified GUI or visual object to have its geometry managed by its parent. Note that an object cannot be made visible (mapped) until it is managed; however, by default most objects are managed at the time when they are created. The panicon object is an exception to this rule.

G.24. xvw_map() — *map an object*

Synopsis

```
void xvw_map(
   xvobject object)
```

Input Arguments

`object`
the object to be mapped

Description

Map the specified GUI or visual object, causing it to be displayed.

G.25. xvw_name() — *get the name of the object*

Synopsis

```
char *xvw_name(
   xvobject object)
```

Input Arguments

`object`
object for which to get the name

Returns

The name associated with the object on success, NULL on failure

Description

Returns the name associated with the object.

G.26. xvw_numchildren() — *get the number of children of an object*

Synopsis

```
int xvw_numchildren(
   xvobject object,
   xvclass  wclass)
```

Input Arguments

`object`
object to for which to check number of children

`wclass`
the desired class of the children, or NULL for all children

Returns

Returns the number of children found

Description

Returns the number of children of the object. If desired, an object class may be specified, so that only children of that class are counted.

G.27. xvw_object() — *get the object associated with a particular widget*

Synopsis

```
xvobject xvw_object(
   Widget widget)
```

Input Arguments

`widget`

widget for which to return the GUI or visual object

Returns

The object associated with the widget on success, NULL on failure

Description

Returns the GUI or visual object associated with the specified widget. If no object is associated with the widget, then an object is created, the widget is assigned to the new object.

G.28. xvw_parent() — *get the parent of an object*

Synopsis

```
xvobject xvw_parent(
   xvobject object)
```

Input Arguments

`object`

the object for which to return the parent.

Returns

The parent of the object on success, NULL on failure

Description

Returns the parent of the specified GUI or visual object.

G.29. xvw_place() — *place an object on the screen*

Synopsis

```
void xvw_place(
   xvobject object,
   xvobject placement,
   int       xoffset,
   int       yoffset)
```

Input Arguments

`object`
the object to be placed

`placement`
object, CURSOR_PLACEMENT, or SCREEN_PLACEMENT that the object should centered around

`xoffset`
additional x placement offset

`yoffset`
additional y placement offset

Description

This utility is used for placing a created object on the display. This is used most often with custom compound objects that are to be popped up after *xvf_run_form()* has already been called.

The object placement variable allows the programmer to place the object relative to another object. The object's placement is calculated to center the object relative to that of the placement object. Alternatively the placement object can be specified as CURSOR_PLACEMENT, which will center the object around the cursor, or SCREEN_PLACEMENT which will center the object in the center of the screen.

The xoffet & yoffset can be used to adjust the offset of the placement. The offset can be either negative or positive, but must be less than the screen dimensions. If the calculation of placed object would cause it to not appear within the screen then the object's placement will be readjusted to automatically fit.

G.30. xvw_raise() — *raise an object*

Synopsis

```
void xvw_raise(
   xvobject object)
```

Input Arguments

`object`
the object to be raised

Description

Raises a GUI or visual object with respect to other siblings in the visual stacking order. If the object overlaps another sibling object, this call will cause the object to obscure the sibling.

G.31. xvw_realize() — *realize an object*

Synopsis

```
void xvw_realize(
   xvobject object)
```

Input Arguments

`object`
the object to be realized

Description

Realizes a GUI or visual object. The realization process involves creation of the actual (displayable) parts of the object, including any window(s) that may be associated with the object.

An object cannot be made visible (mapped) until it is realized; however, by default most objects are realized at the time when they are created.

G.32. xvw_refresh() — *refreshes an object*

Synopsis

```
void xvw_refresh(
   xvobject object)
```

Input Arguments

`object`
the object to be refreshed

Description

Refreshes a GUI or visual object.

G.33. xvw_rootwindow() — *get the root window associated with an object*

Synopsis

```
Window xvw_rootwindow(
   xvobject object)
```

Input Arguments

`object`

object to get the rootwindow from

Returns

The rootwindow associated with the object on success, NULL on failure

Description

Returns the root window associated with the display on which the GUI or visual object appears. If the object is passed in as NULL, then it returns the rootwindow of the default display.

G.34. xvw_sensitive() — *sensitize or de-sensitize an object*

Synopsis

```
void xvw_sensitive(
   xvobject object,
   int    sensitive)
```

Input Arguments

`object`

the object to be affected

`sensitive`

FALSE if object is to be de-sensitized. TRUE if object is to be re-sensitized,

Description

Sensitizes or de-sensitizes a GUI or visual object. When an object is made insensitive, it will ignore all attempts at user input; it will also appear "dim" or "stippled" to indicate its insensitive state (a current limitation restricts this capability to widgets only; insensitive gadgets will not accept input, but they cannot provide a visual cue to reflect their insensitive state). All objects are normally sensitive.

G.35. xvw_screen() — *return the screen associated with a object*

Synopsis

```
Screen *xvw_screen(
   xvobject object)
```

Input Arguments

`object`

object for which to get the screen

Returns

The Screen associated with the object on success, NULL on failure

Description

Returns the display screen associated with the object. Note that this is the GUI & Visualization services equivalent of *XtScreen()*. If object is NULL, returns the default screen.

Restrictions

Restrictions on data or input as applicable

G.36. xvw_screennum() — *return the screen number associated with an object*

Synopsis

```
int xvw_screennum(
   xvobject object)
```

Input Arguments

`object`

object for which to get the screen number

Returns

The screen number associated with the object on success, -1 on failure

Description

Returns the screen index number associated with the object. Note that this is the GUI & Visualization services equivalent of *XScreenNumberOfScreen()*. If object is NULL, returns the default screen number.

G.37. xvw_sort() — *sort a list of objects*

Synopsis

```
int xvw_sort(

  xvobject *object,
  int      num)
```

Input Arguments

`num`

the number of objects in the array

Returns

TRUE (1) on success, FALSE (0) on failure

Description

This routine will sort a list of GUI and visual objects, first by their Y position, and then by their X position.

G.38. xvw_toplevel() — *get the toplevel object of an object*

Synopsis

```
xvobject xvw_toplevel(
    xvobject object)
```

Input Arguments

`object`

object for which to retrieve the toplevel

Returns

The toplevel associated with the object on success, NULL on failure

Description

Returns the toplevel object associated with a particular with an object.

G.39. xvw_unmanage() — *unmanage an object*

Synopsis

```
void xvw_unmanage(
    xvobject object)
```

Input Arguments

`object`

the object to be unmanaged

Description

Causes the specified GUI or visual object to have its geometry unmanaged by its parent. Unmanaging an unmapped object means that it does not use resources of the X server, freeing them for use by other objects. Furthermore, if the unmapped object does not require management by its parent, the calling application will be able to manage currently mapped objects more efficiently. Note that an object cannot be made visible if it is not managed.

G.40. xvw_unrealize() — *un-realize an object*

Synopsis

```
void xvw_unrealize(
   xvobject object)
```

Input Arguments

`object`

the object to be un-realized

Description

When an object is *realized*, it has its windows created on the display, and its final initializations done. This routine allows you to un-realize an object. This destroys the windows associated with an object and its descentants, but does not destroy the instance of the object itself; the object can be still be re-realized and used again at a later date.

G.41. xvw_unmap() — *unmap an object*

Synopsis

```
void xvw_unmap(
   xvobject object)
```

Input Arguments

`object`

the object to be unmapped

Description

Unmap the GUI or visual object specified, causing it to "disappear".

G.42. xvw_visual() — *get the visual associated with an object*

Synopsis

```
Visual *xvw_visual(
   xvobject object)
```

Input Arguments

`object`
object to get the visual from

Returns

The default visual associated with the object or gadget

Description

Returns the visual associated with the object. If the object is NULL then it returns the visual associated with the default display.

G.43. xvw_widget() — *get the widget (or gadget) associated with an object*

Synopsis

```
Widget xvw_widget(
   xvobject object)
```

Input Arguments

`object`
object for which to return the widget (or gadget)

Returns

The widget (or gadget) associated with the GUI or visual object on success, NULL on failure

Description

Returns the widget (or gadget) associated with the specified GUI or visual object.

G.44. xvw_window() — *get the window associated with an object*

Synopsis

```
Window xvw_window(
   xvobject object)
```

Input Arguments

`object`
: object to get the window

Returns

The window associated with the object or gadget on success, KNONE on failure

Description

Returns the window associated with a GUI or visual object.

H. The Button Object

Figure 1: The button GUI object, instantiated as an "Edit" button.

H.1. xvw_create_button() — *create a button object*

Synopsis

```
xvobject xvw_create_button(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: parent of the button object

`name`
: a name for this particular instance of the button object (for use in app-defaults files, etc)

Returns

The GUI button object on success, NULL on failure.

Description

A button object is a rectangular or oval mechanism displaying either a text label or a pixmap. If a callback is installed on the button, software control will be passed to the callback routine when the user clicks the mouse on the button.

H.2. Attributes of the Button Object

Summary of Button Attributes

Attribute	Description
XVW_ATEXT	The accelerator text used for the button object. This is used when there is an accelerator/action associated with a button.
XVW_BUTTON_ARM	The arm callback is called when the button is about to be selected or deselected. When the user performs a button press the button is said to be "armed", when they release the button will generate a button select callback, which will be followed by a dis-arm callback. xvw_add_callback(button, XVW_BUTTON_ARM, button_cb, client_data);
XVW_BUTTON_SELECT	A button is not much fun without a callback installed on it; it has no functionality otherwise. Use *xvw_add_callback()* to install a callback on the button object which will be fired when the user clicks on the button. When calling *xvw_add_callback()*, pass this attribute directly, as in `xvw_add_callback(button, XVW_BUTTON_SELECT, button_cb, client_data);`
XVW_BUTTON_SELECTED	Whether the button is selected or not, this will give the appearance that the button is depressed or not. Which can be used by certain applications to indicate be used to show optional selected buttons.
XVW_LABEL	The label used for the button object. Note that this is only used when the label is not NULL, otherwise if the label is NULL, then no text will appear in the button.
XVW_LABEL_JUSTIFY	Specifies whether the label on the button is to be left justified, middle justified, or right justified. Note that this is only used when the XVW_LABEL is set not set to NULL.

Summary of Button Attributes	
Attribute	**Description**
XVW_PIXMAP	This is the pixmap (or bitmap) that appears on the button when the XVW_PIXMAP. is not set to NULL. If NULL (the default) then no pixmap will appear within the button. Candidates for the value of this attribute may be created with the use of *XCreatePixmap()*; see *The Xlib Reference Manual* by O'Reilly and Associates. Note that this attribute is mutually exclusive with XVW_PIXMAP_FILENAME; specify one or the other, not both.
XVW_PIXMAP_FILENAME	This is the name of the file defining the pixmap (or bitmap) to be displayed.
XVW_PIXMAP_JUSTIFY	Specifies how the pixmap is justified within the pixmap object. The pixmap can be set to be left justified, middle justified, or right justified. Note that the same values are used as for XVW_LABEL_JUSTIFY.

Descriptions of Button Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_ATEXT (atext)	char *	NULL	any printable text
XVW_BUTTON_ARM (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_BUTTON_SELECT (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_BUTTON_SELECTED (N/A)	int	FALSE	TRUE/FALSE
XVW_LABEL (label)	char *	NULL	any printable text
XVW_LABEL_JUSTIFY (labelJustify)	int	KLABEL_JUSTIFY_CENTER	KLABEL_JUSTIFY_LEFT KLABEL_JUSTIFY_CENTER KLABEL_JUSTIFY_RIGHT
XVW_PIXMAP (pixmap)	Pixmap	NULL	Valid Pixmap structure
XVW_PIXMAP_FILENAME (N/A)	char *	NULL	The full path to a valid xpm or xbm file, dfining the desired pixmap (or bitmap). Note that the path may contain $TOOLBOX.

Descriptions of Button Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_PIXMAP_JUSTIFY (pixmapJustify)	int	KPIXMAP_JUSTIFY_CENTER	KPIXMAP_JUSTIFY_LEFT KPIXMAP_JUSTIFY_CENTER KPIXMAP_JUSTIFY_RIGHT

H.3. About Callbacks on Buttons

When the user clicks on the button, software control is passed to the installed callback. The callback to which the button object will pass control must be installed on the button object by the application after the button object is created, with a call of the composition:

```
xvw_add_callback(button_object, XVW_BUTTON_SELECT,
                 callback, client_data);
```

The callback routine must be defined according to the syntax:

```
void callback(
   xvobject object,
   kaddr  client_data,
   kaddr  call_data)
{
     /* do something in response to the button click */
}
```

I. The Label Object

Figure 2: The label GUI object is used to display text in Cantata.

I.1. xvw_create_label() — *create a label object*

Synopsis

```
xvobject xvw_create_label(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: parent of the label object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The label object on success, NULL on failure

Description

A label object is for the display of non-editable text on VisiQuest graphical user interfaces. If desired, a label object may also display a pixmap, in addition to the text. The label object supports multiline text; simply include "\n" in the text to indicate the end of a line. Label text and pixmaps may be left, center, or right justified.

There are three major differences between a *label* object and a *text* object. Firstly, a label object is a gadget; ie, it has no window of its own, but uses the window of its parent. A text object, on the other hand, is a widget; it has a dedicated window of its own. Secondly, the label object supports the display of non-editable text only; the text displayed by a text object can be edited by the user unless it is explicitly handled by the application via the paste callback, irregardless if a cursor is displayed. Thirdly, the label object is implemented as part of VisiQuest GUI services; the text object, use the label as a subpart in order to provide editability. For most non-editable text display uses, a label object is preferable, and has the advantage that it is much faster to refresh than a text object.

The difference between a *label* object and a *string* object is that the label object is meant for display of text on graphical user interfaces, while the string object is an annotation, meant for display of text on images, area objects, and so on.

I.2. Attributes of the Label Object

Summary of Label Attributes	
Attribute	**Description**
XVW_FORCE_REDISPLAY	This attribute of the manager gadget is inherited by the label object; it is emphasized here because it can be used by the label object to achieve smoother, faster update of the label. It is especially useful when the label is updated very rapidly, as for example in an event handler installed on button motion.
XVW_LABEL	The label used for the label object.
XVW_LABEL_EMPHASIZE	This attribute causes the label to be drawn twice, giving it a 3D, emphasized effect.
XVW_LABEL_EMPHASIZECOLOR	When XVW_LABEL_EMPHASIZE is set to TRUE, this is the color in which the label is emphasized.
XVW_LABEL_EMPHASIZEPIXEL	When XVW_LABEL_EMPHASIZE is set to TRUE, this is the pixel value associated with the color in which the label is emphasized.
XVW_LABEL_JUSTIFY	Specifies whether the label on the label object is to be left justified, middle justified, or right justified.

Descriptions of Label Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_FORCE_REDISPLAY (forceRedisplay)	int	FALSE	TRUE/FALSE
XVW_LABEL (label)	char *	NULL	any printable text
XVW_LABEL_EMPHASIZE (labelEmphasize)	int	FALSE	TRUE/FALSE
XVW_LABEL_EMPHASIZECOLOR (labelEmphasizeColor)	char *	the background color	any valid color name: see /usr/lib/X11/rgb.txt or Section 7.1.1 of *The XLib Programming Manual* by Adrian Nye
XVW_LABEL_EMPHASIZEPIXEL (N/A)	unsigned long	pixel associated with the background color	any valid pixel value: see Section 7.3 and 7.4 of *The XLib Programming Manual* by Adrian Nye
XVW_LABEL_JUSTIFY (labelJustify)	int	KLABEL_JUSTIFY_CENTER	KLABEL_JUSTIFY_LEFT KLABEL_JUSTIFY_CENTER KLABEL_JUSTIFY_RIGHT

I.3. Button & Label Example

```
#include <design.h>


/*
 *   This example program puts up 5 pairs of labels & buttons.
 *   It adds very simple callbacks to the buttons.
 */
#define BUTTON_NUM 5

static void button_cb PROTO((xvobject, kaddr, kaddr));

void main(
   int  argc,
   char *argv[])
{
     int      i;
     int      *button_id;
     char     temp[KLENGTH];
     xvobject manager;
     xvobject button;
     xvobject label;
     xvobject horiz_offset;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidget library */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror("example", "main", "Cannot open display");
        kexit(KEXIT_FAILURE);
     }

     /* create a manager backplane */
     manager = xvw_create_manager(NULL, "back");

     /*
      * create 5 pairs of buttons & labels, each button appearing
      * under its label.  Using horiz_offset allows us to create the
      * pairs in a line.
      */
     horiz_offset = NULL;
     for (i = 0; i < BUTTON_NUM; i++)
     {
        ksprintf(temp, "label%d", i+1);
        label  = xvw_create_label(manager, "label");
        xvw_set_attributes(label,
                XVW_LABEL,        temp,          /* numbered label  */
                XVW_RIGHT_OF,     horiz_offset,  /* line of buttons */
                XVW_BELOW,        NULL,          /* top row         */
                XVW_CHAR_WIDTH,   10.0,          /* 10 chars wide   */
                XVW_CHAR_HEIGHT,  2.0,           /* 2 char  high    */
                XVW_BORDER_WIDTH, 0,             /* no border       */
                NULL);
```

```
        ksprintf(temp, "button%d", i+1);
        button = xvw_create_button(manager, "button");
        xvw_set_attributes(button,
                XVW_LABEL,        temp,          /* numbered label  */
                XVW_RIGHT_OF,     horiz_offset,  /* line of labels  */
                XVW_BELOW,        label,         /* below the label */
                XVW_CHAR_WIDTH,   10.0,          /* 10 chars wide   */
                XVW_CHAR_HEIGHT,  2.0,           /* 2 char high     */
                XVW_BORDER_WIDTH, 1,             /* thin border     */
                NULL);

        button_id = (int *) kmalloc(sizeof(int));
        *button_id = 100+i;
        xvw_add_callback(button, XVW_BUTTON_SELECT, button_cb, button_id);

        horiz_offset = label;
     }

     /* display & run program */
     xvf_run_form();
}

static void button_cb (
    xvobject object,
    kaddr    client_data, /* not used */
    kaddr    call_data)   /* not used */
{
     char *label;

     /*
      * sent in a pointer to an integer as client_data;
      * have to retrieve it by using an appropriate cast
      */
     int  *number = (int *) client_data;

     /*
      *  see what the label of the button is
      */
     xvw_get_attribute(object, XVW_LABEL, &label);

     /*
      *  print out button label, with ID number
      */
     kfprintf(kstderr, "Button click on %s, ID number %d\n",
             label, *number);
}
```

J. The List Object

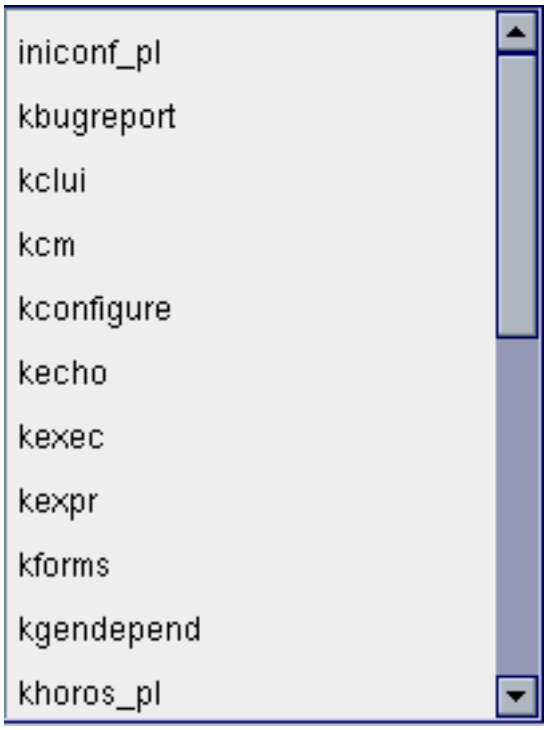


Figure 3: The list object is used by craftsman to list software objects in a toolbox.

J.1. xvw_create_list() — *create a list object*

Synopsis

```
xvobject xvw_create_list(

    xvobject parent,
    char   *name)
```

Input Arguments

`parent`
: parent of the list object

`name`
: a name for this particular instance of the list object (for use in app-defaults files, etc)

Returns

The list object on success, NULL on failure

Description

The *list* GUI object presents the user with a set of strings inside a scrolled viewport. A callback can be installed on the list object which will be fired when the user selects an item from the list.

A list GUI object is really a *compound* object; that is, it is made up of two objects: the viewport and the "actual list", where the viewport contains the list and provides scrolling capabilities, and the "actual list" is inside the viewport and contains the list contents.

When you create a list, the GUI object returned is the viewport object; this object should be referenced when setting geometry, relative offset, and so on (in other words, when setting or getting any attribute not having to do *directly* with the contents of the list, where those particular attributes are called out below).

The "actual list" part of the object is used when changing the contents of the list, setting the size of the list, and highlighting or unhighlighting elements of the list. The "actual list" part of the compound list object can be obtained with a call to xvw_retrieve_list().

When the Athena widget set is used, the list object is instantiated as a xfwfScrolledListWidget. When the Olit widget set is used, the list object is instantiated as a scrollingListWidget. When the Motif widget set is used, the list object is instantiated as a ScrolledList.

J.2. Attributes of the List Object

Summary of List Attributes

Attribute	Description
XVW_LIST_ADD	This action attribute adds the specified string to the end of the list and automatically increments XVW_LIST_SIZE. The displayed list is automatically updated to display the new string.
XVW_LIST_DELETE	This action attribute deletes the specified string from the list and automatically decrements XVW_LIST_SIZE. The displayed list is automatically updated so that the old string is no longer displayed.
XVW_LIST_DELETEALL	This action attribute deletes all strings from the list and automatically sets XVW_LIST_SIZE to 0 and XVW_LIST_INDEX to -1. The displayed list is emptied.
XVW_LIST_HIGHLT_ELEM	This *action* attribute may be used to highlight an element in the list. Provide the index of the element to be highlighted. If the element is already highlighted when the action attribute is used, nothing will happen. Remember that indexing begins at 0.
XVW_LIST_INDEX	This attribute specifies the selected item in the list by index. Getting this attribute returns the index of the selected item in the list, or -1 if no item is selected. Setting this attribute causes the item with the specified index to be selected, or if set to -1 causes the list to have no item selected .

Summary of List Attributes	
Attribute	**Description**

XVW_LIST_ITEM_ACTION	*xvw_add_callback()* may be used to install a callback on the list object which will be fired when the user clicks *twice, rapidly* on an item in the list.
	
	When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(listobj,`
	`XVW_LIST_ITEM_ACTION,`
	`list_cb, client_data);`

XVW_LIST_ITEM_SELECT	*xvw_add_callback()* may be used to install a callback on the list object which will be fired when the user clicks *once* on an item in the list.
	
	When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(listobj,`
	`XVW_LIST_ITEM_SELECT,`
	`list_cb, client_data);`

XVW_LIST_SIZE	This attribute specifies size of the array of strings that are displayed by the list object. Note that when setting this attribute, you should set the XVW_LIST_STRINGS attribute at the same time to indicate the strings in the array!
XVW_LIST_STRING	This attribute specifies the selected item in the list by string. Getting this attribute returns the string corresponding to the selected item of the list, or NULL if no item is selected. Setting this attribute causes the selected item in the list to have its string updated to this new value. Setting this attribute to NULL causes the list to have no item selected.
XVW_LIST_STRINGS	This attribute is the array of strings representing the items that are displayed by the list object. Note that when setting this attribute, you MUST set the XVW_LIST_SIZE attribute at the same time to indicate the number of strings in the array! Similarly, if getting this attribute, you should get the XVW_LIST_SIZE attribute as well to tell you the number of strings in the returned array.
XVW_LIST_UNHIGHLT_ELEM	This *action* attribute may be used to un-highlight an element in the list. Provide the index of the element to be un-highlighted. If the element is not highlighted when the action attribute is used, nothing will happen. Remember that indexing begins at 0.

Descriptions of List Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_LIST_ADD (N/A)	char *	N/A	the string to be added to the list

Descriptions of List Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_LIST_DELETE (N/A)	char *	N/A	the string to be deleted from the list
XVW_LIST_DELETEALL (N/A)	int	N/A	TRUE
XVW_LIST_HIGHLT_ELEM (N/A)	int	N/A	0 <= value <= XVW_LIST_SIZE -1
XVW_LIST_INDEX (N/A)	int	-1	-1 for no string selected; otherwise, the index of the string that is selected (0 to XVW_LIST_SIZE - 1)
XVW_LIST_ITEM_ACTION (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_LIST_ITEM_SELECT (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_LIST_SIZE (N/A)	int	0	size of the string array being displayed
XVW_LIST_STRING (N/A)	kstring	NULL	NULL for no item selected; otherwise, the string of the item that is selected
XVW_LIST_STRINGS (N/A)	kstring *	NULL	Array of strings, of size specified by XVW_LIST_SIZE.
XVW_LIST_UNHIGHLT_ELEM (N/A)	int	N/A	0 <= value <= XVW_LIST_SIZE -1

J.3. About Callbacks on Lists

When the user selects a string from the list object, software control is passed to the installed callback. The callback to which the list object will pass control must be installed on the compound list object (as opposed to the "actual list" object) by the application after the list object is created, with a call of the composition:

```
xvw_add_callback(list_object, XVW_LIST_ITEM_SELECT,
           callback, client_data);
```

The callback routine must be defined according to the syntax:

```
void callback(
```

```
   xvobject list_object,
   kaddr  client_data,
   kaddr  call_data)
{
     xvw_list_struct *list_return;

     /* see "USING CALLBACKS" for use of client_data */
     /* cast the pointer to the call_data */

     list_return = (xvw_list_struct *) call_data;
     /* now, list_return->string holds the selected string */
     /* now, list_return->list_index has the index of the selected string */
     :
     :
}
```

As implied in the above specification, user's click on one of the items in the list causes the *xvwidgets* library to pass software control to the callback routine defined. The callback routine will be called with the call_data containing the current string value and its index stored in following structure :

```
/*
 *  The list structure returned as call_data from the list object
 */
typedef struct _xvw_list_struct {
        char   *string;   /* the string selected by the user */
        int    list_index;     /* index of the string */
} xvw_list_struct;
```

J.4. List Example

```
#include <design.h>


/*
 *   This program puts up two list objects; it emphasizes the compound nature
 *   of the list object which consists of a scrolled backplane and the actual
 *   list object.
 *
 *   The xvw_change_list() routine is used to set the lists' contents;
 *   passing different values for the last argument to xvw_change_list
 *   determine whether a scrollbar will appear.
 *
 *   A very simple callback is added to the list items, which prints out the
 *   index and string associated with a list item when the user selects it.
 *
 *   This example also serves to illustrate the use of call_data.
 *
 */
void list_item_cb PROTO((xvobject, kaddr, kaddr));
void quit_cb      PROTO((xvobject, kaddr, kaddr));

void main(
   int  argc,
   char *argv[])
{
```

```
   xvobject manager;
   xvobject button;
int      num1,  num2;
   xvobject list1, list2, actual_list;

   static char *geographical_strings[] = {
           "mountains", "rivers", "oceans", "plains",
           "forests", "deserts", "marshes", "reefs",
           "glaciers", "jungles"
   };

   static char *animal_strings[] = {
     "bears", "cats", "fish", "cockroaches", "birds",
     "spiders", "dogs", "lizards", "frogs",
     "tigers", "dolphins", "rabbits", "mooses",
     "monkeys", "antelopes"
   };

/*
 * "knumber" is a convenient routine that counts the number of items
    *  in a static array of strings
    */
   num1 = knumber(geographical_strings);
   num2 = knumber(animal_strings);

   /* initialize VisiQuest program */
   khoros_initialize(argc, argv, "DESIGN");

/* initialize the xvwidget library */
   if (!xvw_initialize(XVW_MENUS_XVFORMS))
   {
      kerror("example", "main", "Cannot open display");
      kexit(KEXIT_FAILURE);
   }

/* create a manager backplane */
   manager = xvw_create_manager(NULL, "back");
   xvw_set_attributes(manager,
                      XVW_RESIZABLE,       TRUE,
                      NULL);

/*
 * create the first list. the list object is a compound object,
    * consisting of:
    *     1 - a scrolled backplane
    *        2 - the actual list
 */
   list1 = xvw_create_list(manager, "list1");
   xvw_set_attributes(list1,
               XVW_RESIZABLE,     TRUE,   /* resizable         */
               XVW_CHAR_WIDTH,    20.0,   /* width of 20 chars */
               XVW_CHAR_HEIGHT,   5.0,    /* height of 5 chars */
               XVW_BELOW,         NULL,   /* at the top        */
               XVW_RIGHT_OF,      NULL,   /* at the left edge  */
               NULL);

/*
 * retrieve the list component of the list object, so that
    * we can set the contents of it, set attributes on it,
```

```
         * and install the callback.
        */
       actual_list = xvw_retrieve_list(list1);

          /*
        * Use xvw_change_list() to set the contents of the list.
           * Since the last parameter on xvw_change_list() is TRUE,
           * XVW_CHAR_WIDTH and XVW_CHAR_HEIGHT will be => over-ridden
           * by the actual width & height of the list.   So, xvw_change_list()
           * is being allowed to resize the list widget to fit the given list;
        * thus, there will NOT be a scroll bar on the first list.
           */
          xvw_change_list(actual_list, geographical_strings, num1, TRUE);
       xvw_set_attribute(actual_list, XVW_LIST_HIGHLT_ELEM, 3);
       xvw_set_attribute(actual_list, XVW_LIST_UNHIGHLT_ELEM, 3);

       /*
        * add the list_item_cb() callback on the list.
        */
          xvw_add_callback(actual_list, XVW_LIST_ITEM_SELECT,
                  list_item_cb, "list1");

       /*
           * Since the last parameter on xvw_change_list() is FALSE, the second
           * list is constrained to the width and height indicated by the
           * values of the XVW_CHAR_WIDTH and XVW_CHAR_HEIGHT attributes.
           * Since the list is too big to fit in the space specified, a
           * scrollbar will appear.
           */
          list2 = xvw_create_list(manager, "list2");
          xvw_set_attributes(list2,
                       XVW_RESIZABLE,     TRUE,   /* resizable         */
                       XVW_RIGHT_OF,      list1,  /* R of the 1st list */
                       XVW_CHAR_WIDTH,    20.0,   /* width of 20 chars */
                       XVW_CHAR_HEIGHT,   5.0,    /* height of 5 chars */
                       XVW_BELOW,         NULL,   /* at the top        */
                       NULL);
       actual_list = xvw_retrieve_list(list2);

          xvw_change_list(actual_list, animal_strings, num2, FALSE);
          xvw_add_callback(actual_list, XVW_LIST_ITEM_SELECT,
                     list_item_cb, "list2");

       /* Add a button to quit the program */
          button = xvw_create_button(manager, "button");
          xvw_set_attributes(button,
                    XVW_LABEL,  "Quit",
                    XVW_BELOW,  list2,
                    XVW_LEFT_OF, NULL,
                    NULL);
          xvw_add_callback(button, XVW_BUTTON_SELECT, quit_cb, NULL);

       /* display and run */
          xvf_run_form();
}

void list_item_cb (
    xvobject object,
    kaddr    client_data,
```

```
    kaddr    call_data)
{
     xvw_list_struct *list_return;
     char  *title = (char *) client_data;

     /*
      * a list object is one of the few objects that uses a call_data
         * structure.  it needs to => return information to the caller,
         * so that information is returned via the call_data.  The data type
         * of the call_data is determined by the object itself;  a list
         * object uses a call_data of type "xvw_list_struct".  The
         * xvw_list_struct data structure has two fields: a char *string,
         * and an int list_index.
      */
     list_return = (xvw_list_struct *) call_data;

     /*
      * use contents of call_data structure to display index & string
      */
     kfprintf(kstderr, "Chosen was item %d of %s, %s\n",
           list_return->list_index, title, list_return->string);

}

void quit_cb (
    xvobject object,
    kaddr    client_data,
    kaddr    call_data)
{
        exit(0);
}
```

K. The Menu & MenuButton Objects

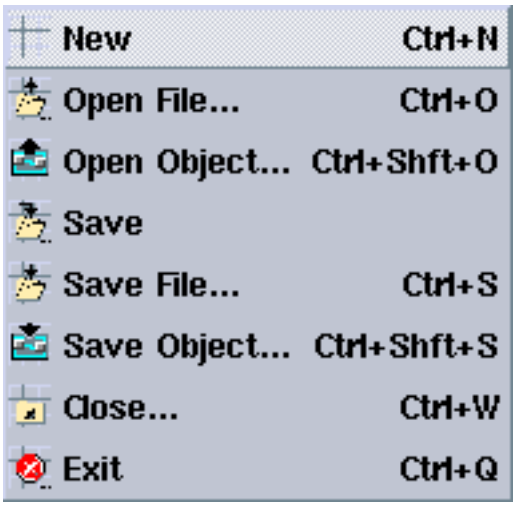


Figure 4: The menubutton GUI object is used to pull down a menu of buttons and labels. Here, the Motif version of a menubutton/menu pair lists names of subforms and action buttons.

K.1. xvw_create_menubutton() — *create a menubutton object*

Synopsis

```
xvobject xvw_create_menubutton(
    xvobject parent,
    char     *name)
```

Input Arguments

`parent`
: parent of the menubutton object

`name`
: a name for this particular instance of the menubutton object (for use in app-defaults files, etc)

Returns

The GUI menubutton object on success, NULL on failure.

Description

A menubutton object is a rectangular or oval mechanism displaying either a text label or a pixmap. If a callback is installed on the menubutton, software control will be passed to the callback routine when the user clicks the mouse on the menubutton.

K.2. xvw_create_menu() — *create a menu object*

Synopsis

```
xvobject xvw_create_menu(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: parent of the menu object

`name`
: a name for this particular instance of the menu object (for use in app-defaults files, etc)

Returns

The menu GUI object on success, NULL on failure.

Description

The *menu* GUI object is the menu which is associated with a button n which the user can click to pull down a menu of items; holding the mouse button down, the user may select an item from the menu. Items in the menu may be *button* objects or *label* objects. Button object items in the menu will be highlighted when the user moves the pointer over them. If the mouse button is released with the pointer over a button in the menu when that button has a callback installed,* the button callback will be fired normally.

It is important to understand that the *menubutton* GUI object is a *compound* object. That is, there are actually two objects created: the menubutton and the menu itself, where the menubutton is the button that appears on the GUI, and the menu is what pops up when the user clicks on the menubutton.

When you create a menubutton object, the GUI object returned is the *button*; this object should be referenced when setting geometry, relative offset, and so on. Attributes such as label, width, and height may also be set on the menubutton.

However, in order to add buttons and labels to the pulldown menu, you will to call *xvw_create_button()* and *xvw_create_label()* with the *menu* (not the *menubutton*) as the parent. The actual menu component of the compound menubutton object can be obtained with: `xvw_get_attribute(menubutton, XVW_MENUBUTTON_MENU, &menu);`

Summary of Menu Attributes

Attribute	Description

Summary of Menu Attributes	
Attribute	**Description**
XVW_MENU_TEAR_OFF	If desired, this allows the user to create a tear-away, where the menu will appear to allow the user to tear off/away the menu into it's toplevel window.

Descriptions of Menu Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_MENU_TEAR_OFF (menuTearOff)	int	FALSE	TRUE/FALSE

Summary of MenuButton Attributes	
Attribute	**Description**
XVW_MENUBUTTON_MENU	This is a read-only attribute that will return the pointer to the xvobject for the menu associated with the menubutton. The menu is actually created by the menubutton object, and this resource.

Descriptions of MenuButton Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_MENUBUTTON_MENU (N/A)	xvobject	NULL	valid xvobject

K.3. MenuButton Example

```
#include <design.h>

/*
 *   This program illustrates the use of the menubutton object; it
 *   emphasizes the compound nature of the menubutton object, which consists
 *   of the menu button itself, plus the submenu which contains the buttons
 *   and/or labels which are displayed when the user clicks on the menubutton.
 */

#define PAIR_NUM 5

void button_cb PROTO((xvobject, kaddr, kaddr));
void quit_cb   PROTO((xvobject, kaddr, kaddr));
```

```
void main(
   int  argc,
   char *argv[])
{
     int      i;
     char     temp[KLENGTH];
     xvobject manager;
     xvobject menubutton;
     xvobject submenu;
     xvobject button;
     xvobject label;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidget library */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror("example", "main", "Cannot open display");
        kexit(KEXIT_FAILURE);
     }

     /* create a manager backplane */
     manager = xvw_create_manager(NULL, "back");
     xvw_set_attributes(manager,
                    XVW_WIDTH,  100,   /* 100 pixels wide */
                    XVW_HEIGHT, 100,   /* 100 pixels high */
                    NULL);

     /*
      *  create the menubutton, which is actually a compound object
         *  consisting of two xvobjects:
         *    1) the menubutton, which is always displayed, and on which the
         *                       user may click to display the submenu
         *    2) the submenu,    which will be the parent to the buttons
         *                       & labels on the menu.
      */
     menubutton = xvw_create_menubutton(manager, "menubutton");

     /*
      *  set the label of the menubutton, and center it in the middle
         *  of the manager backplane.
      */
     xvw_set_attributes(menubutton,
             XVW_LABEL,   "Menu Button", /* Set label.               */
             XVW_ABOVE,    NULL,      /* By setting all the          */
             XVW_BELOW,    NULL,      /* relative positioning        */
             XVW_LEFT_OF,  NULL,      /* attributes to NULL, put the */
             XVW_RIGHT_OF, NULL,      /* button in middle of manager */
             NULL);

     /*
      *  once the menubutton is created, you can obtain the submenu
         *  with the function named appropriately.
      */
     submenu = xvw_retrieve_menu(menubutton);

     /*
      *  want to have 5 pairs of buttons & labels appear on the submenu.
```

```
         *  create the buttons and labels, using submenu as the parent.
        */
       for (i = 0; i < PAIR_NUM; i++)
       {
          /*
              *  Alternate buttons with labels on the menu.  Both the labels and
           *  buttons are labelled incrementally;  on a menu, each item will
              *  automatically be placed beneath the last, there is no need to
              *  use positioning attributes (in fact, they are ignored).
              *  Install the "button_cb" callback on the buttons.
              */
          sprintf(temp, "button%d", i+1);
          button = xvw_create_button(submenu, "button");
          xvw_set_attribute(button, XVW_LABEL,  temp);
          xvw_add_callback(button, XVW_BUTTON_SELECT,
                     button_cb, NULL);

          sprintf(temp, "label%d", i+1);
          label  = xvw_create_label(submenu, "label");
          xvw_set_attribute(label, XVW_LABEL,  temp);
       }

       /*
        *  last, put in a button with a callback that will allow
           *  the user to quit the program.
        */
       button = xvw_create_button(submenu, "button");
          xvw_set_attribute(button, XVW_LABEL,  "Quit");
          xvw_add_callback(button, XVW_BUTTON_SELECT,
                           quit_cb, NULL);

       /* display and run the program */
       xvf_run_form();
}

void button_cb (
    xvobject object,
    kaddr    client_data,
    kaddr    call_data)
{
     char *label;

          xvw_get_attribute(object, XVW_LABEL, &label);
          fprintf(stderr, "Selection of '%s'\n", label);
}

void quit_cb (
    xvobject object,
    kaddr    client_data,
    kaddr    call_data)
{
     exit(0);
}
```

K.4. About Callbacks on Menubuttons

It is not necessary to add a callback to the menubutton; by its nature, it will display the pulldown menu when it receives a button press event. Callbacks can (and should) be added to each button that is created as part of the menu, but these callbacks are identical to callbacks on any other button, see the section on "Button GUI Object."

L. The Pixmap Object

Figure 5: The label GUI object can also be used to display a pixmap.

L.1. xvw_create_pixmap() — *create a pixmap object*

Synopsis

```
xvobject xvw_create_pixmap(
   xvobject parent,
   char   *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The pixmap object on success, NULL on failure

Description

A pixmap object simply supports display of a bitmap or pixmap, as defined by an xbm or xpm file.

L.2. Attributes of the Pixmap Object

Summary of Pixmap Attributes	
Attribute	**Description**
XVW_PIXMAP	This is the pixmap (or bitmap) that appears in the pixmap object. Candidates for the value of this attribute may be created with the use of *XCreatePixmap()*; see *The Xlib Reference Manual* by O'Reilly and Associates. Note that this attribute is mutually exclusive with XVW_PIXMAP_FILENAME; specify one or the other, not both.
XVW_PIXMAP_FILENAME	This is the name of the file defining the pixmap.
XVW_PIXMAP_FORCED_HEIGHT	This will force a pixmap to take on the specified height. The pixmap will be subsampled or supersampled as needed.
XVW_PIXMAP_FORCED_WIDTH	This will force a pixmap to take on the specified width. The pixmap will be subsampled or supersampled as needed.
XVW_PIXMAP_JUSTIFY	Specifies how the pixmap is justified within the pixmap object. The pixmap can be set to be left justified, middle justified, or right justified. Note that the same values are used as for XVW_LABEL_JUSTIFY.
XVW_PIXMAP_MASK	Pixmaps are defined by a rectangular grid of values. For implementing non-rectangular pixmaps, a bitmap is used to indicate which portions of the rectangular pixmap should appear, and which portions should not be displayed. This bitmap is referred to as a *mask*. The bitmap must be defined in an xbm file, and the bitmap should be of the same size as the pixmap being displayed. Anywhere a 1 appears in the bitmap, the value of the pixmap at that point is displayed; anywhere a 0 appears in the bitmap, the value of the pixmap at that point will not be displayed. Thus, a circular pixmap can be implemented with a pixmap/bitmap mask pair: the pixmap will define the picture in the circle, while the bitmap mask has all 1's within the boundaries of the circle, and all 0's outside the circle bounds. Candidates for the bitmap may be created with *XCreateBitmapFromData()*; see *The Xlib Reference Manual* by O'Reilly and Associates. Note that this attribute is mutually exclusive with XVW_PIXMAP_MASKNAME; specify one or the other, not both.
XVW_PIXMAP_MASKNAME	This is the file defining the bitmap mask used with non-rectangular pixmaps; see XVW_PIXMAP_MASK for more details.

Descriptions of Pixmap Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_PIXMAP (pixmap)	Pixmap	NULL	Valid Pixmap structure

Descriptions of Pixmap Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_PIXMAP_FILENAME (pixmapFilename)	char *	NULL	The full path to a valid xpm or xbm file, defining the desired pixmap. Note that the path may contain $TOOLBOX.
XVW_PIXMAP_FORCED_HEIGHT (pixmapForcedWidth)	int	-1	Any positive integer. A value of -1 denotes that that this attribute should be ignored.
XVW_PIXMAP_FORCED_WIDTH (pixmapForcedWidth)	int	-1	Any positive integer. A value of -1 denotes that that this attribute should be ignored.
XVW_PIXMAP_JUSTIFY (pixmapJustify)	int	KPIXMAP_JUSTIFY_CENTER	KPIXMAP_JUSTIFY_LEFT KPIXMAP_JUSTIFY_CENTER KPIXMAP_JUSTIFY_RIGHT
XVW_PIXMAP_MASK (N/A)	Pixmap	NULL	Valid Pixmap structure
XVW_PIXMAP_MASKNAME (pixmapMaskname)	char *	NULL	The full path to the xbm file defining the mask; Note that the path may contain $TOOLBOX.

L.3. Complete Resource Set of the Pixmap Object

The complete resource set for the pixmap object includes:

1. The pixmap object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Section C.2, "Attributes of the VisiQuest 2001 Manager Object".

3. The general object attributes, given in Chapter 2, "The xvwidgets Library", Section B, "General Attributes of GUI and Visual Objects".

L.4. Example using the Pixmap Object

An example program using the pixmap object can be found in `$DESIGN/examples/xvwidgets/pixmap/example.c`.

M. The Rowcol Object

Figure 6: The RowCol GUI object provides a layout area in which visual and gui objects are layed out in a Row/Column fashion. Here, eleven menubuton objects are layed out two abreast, except for the eleventh one.

M.1. xvw_create_rowcol() — *create a row-col object*

Synopsis

```
xvobject xvw_create_rowcol(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent of the rowcol object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The rowcol object on success, NULL on failure

Description

A rowcol object is designed specifically for doing convenient layout of other objects in a table-like format, in rows and columns. The rowcol object may be used as a parent for any GUI or visual object, but works particularly well when laying out objects of similar size and nature, such as labels, buttons, pixmaps, annoations, and so on.

M.2. Attributes of the RowCol Object

Summary of RowCol Attributes	
Attribute	**Description**
XVW_ROWCOL_ACTIVE	This constraint attribute can be placed on any RowCol child. It is used to indicate which children should be actively used in the row/column layout method.
XVW_ROWCOL_NUMBER_ACROSS	This attribute specifies how many area objects will be placed on the same row before a new row is started.
XVW_ROWCOL_PACKING	This attribute specifies how the items within the rowcol objects is to be packed. The default is to pack the items within the RowCol so that they are aligned both vertically and horizontally, AccuSoftOW-COL_PACK_ROWCOL. AccuSoftOWCOL_PACK_ROW: packs only along the rows (vertically). AccuSoftOWCOL_PACK_COLUMN: packs only along the columns (horizontally). AccuSoftOWCOL_PACK_ROWCOL: packs along both the rows and columns. AccuSoftOWCOL_PACK_TIGHT: does no packing and will layout
XVW_ROWCOL_SPACING	This attribute specifies the minimum spacing between each objects, in pixels. The default is -1, which means it uses the Manager Object subclass attribute XVW_CANVAS_GRIDSIZE. If the XVW_CANVAS_GRID-SIZE is 0, then the XVW_XSNAP and the XVW_YSNAP is used. If a 0 or positive value is provided, then that is used as the spacing.

Descriptions of RowCol Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_ROWCOL_ACTIVE (rowcolActive)	int	TRUE	TRUE/FALSE
XVW_ROWCOL_NUMBER_ACROSS (rowcolNumberAcross)	int	-1	integer value
XVW_ROWCOL_PACKING (rowcolPacking)	int	AccuSoftOW-COL_PACK_ROWCOL	AccuSoftOWCOL_PACK_ROW AccuSoftOWCOL_PACK_COLUMN AccuSoftOWCOL_PACK_ROWCOL AccuSoftOWCOL_PACK_TIGHT
XVW_ROWCOL_SPACING (rowcolSpacing)	int	-1	integer value

M.3. Complete Resource Set of the RowCol Object

The complete resource set for the rowcol object includes:

1. The rowcol object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Section C.2, "Attributes of the VisiQuest 2001 Manager Object".

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

M.4. Example using the RowCol Object

An example program using the rowcol object can be found in `$DESIGN/examples/xvwidgets/rowcol/example.c`.

N. The Scrollbar Object

Figure 7: The scrollbar is frequently used in VisiQuest GUI's.

N.1. xvw_create_scrollbar() — *create a scrollbar object*

Synopsis

```
xvobject xvw_create_scrollbar(

    xvobject parent,
    char   *name)
```

Input Arguments

`parent`
: parent of the scrollbar object

`name`
: a name for this particular instance of the scrollbar object (for use in app-defaults files, etc)

Returns

The Scrollbar object on success, NULL on failure.

Description

The *scrollbar* GUI object allows users to reposition data that is too large to fit in a viewing window. Typically, the scrollbar is used by other objects that serve as a backplane for a large number of subordinate objects that will not fit within the viewing area. For example, the TextDisplay object has a built-in scrollbar which allows the user to scroll through the text being displayed, and the List object will display a scrollbar if the number of items in the list exceeds the space provided by the height of the List object. The Viewport object also has built-in scrollbars to allow the user to shift the area of view, as does the Canvas object.

The Scrollbar consists of a long, thin area called the scrollbar trough, within which is a smaller, movable button called the slider. Scrollbars may be horizontal or vertical. Horizontal scrollbars have the minimum value at the top and the maximum value at the bottom; vertical scrollbars have the minimum value at the left and the maximum value at the right.

With a horizontal scrollbar, clicking in the scrollbar trough to the right of the slider will cause the scrollbar to increment in value; clicking in the scrollbar trough to the left of the slider will cause the

scrollbar to decrement in value. Similarly with a vertical scrollbar, clicking above the slider will decrement the value while clicking below the slider will increment the value. The slider itself can be dragged back and forth (with a horizontal scrollbar) or up and down (with a vertical scrollbar) to change the value.

A callback can be installed on the scrollbar for incremental movement; this callbeack will be fired when the user clicks in the scrollbar trough (after the scrollbar value is incremented or decremented). The same callback or another callback may be installed on the scrollbar for continuous movement; this callback will be fired when the user moves the slider of the scrollbar (after the scrollbar value is updated).

N.2. Attributes of the Scrollbar Object

Summary of Scrollbar Attributes

Attribute	Description
XVW_ORIENTATION	Indicates whether the scrollbar is to be horizontal or vertical. When a scrollbar is horizontal, the minimum value is on the left and the maximum value is on the right. When a scrollbar is vertical, the minimum value is at the top and the maximum value is at the bottom.

XVW_SCROLL_CONT_MOTION	If desired, *xvw_add_callback()* may be used to install a callback on the scrollbar object which will be fired when the user employs the middle mouse button to move the scrollbar slider continuously. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(scrollbar_obj,`
	`XVW_SCROLL_CONT_MOTION,`
	`scrollbar_cb, client_data);`

XVW_SCROLL_INCR	The value by which the scrollbar slider will be incremented or decremented when the user clicks in the scrollbar trough.

XVW_SCROLL_INCR_MOTION	If desired, *xvw_add_callback()* may be used to install a callback on the scrollbar object which will be fired when the user employs the right or left mouse buttons in the scrollbar trough to move the scrollbar slider incrementally. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(scrollbar_obj,`
	`XVW_SCROLL_INCR_MOTION,`
	`scrollbar_cb, client_data);`

XVW_SCROLL_MAX	The maximum world coordinate value that will be represented by the scrollbar.
XVW_SCROLL_MIN	The minimum world coordinate value that will be represented by the scrollbar.

Summary of Scrollbar Attributes	
Attribute	**Description**
XVW_SCROLL_VALUE	The current (and default) world coordinate represented by the current position of the scrollbar slider.

Descriptions of Scrollbar Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_ORIENTATION (orientation)	int	KSCROLLBAR_ORIENT_HORIZ	KSCROLLBAR_ORIENT_HORIZ KSCROLLBAR_ORIENT_VERT
XVW_SCROLL_CONT_MOTION (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_SCROLL_INCR (N/A)	double	calculated: (scroll max - scroll min)/10	values < (scroll max - scroll min)/2
XVW_SCROLL_INCR_MOTION (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_SCROLL_MAX (N/A)	double	0	value > scroll min
XVW_SCROLL_MIN (N/A)	double	0	value < scroll max
XVW_SCROLL_VALUE (N/A)	double	0	scroll min < value < scroll max

N.3. About Callbacks on Scrollbars

When the user moves the thumb of the scrollbar, software control is passed to the installed callback. The callback may be installed for incremental motion of the scrollbar (XVW_SCROLL_INCR_MOTION), or continuous motion of the scrollbar (XVW_SCROLL_CONT_MOTION), or both. The callback to which the scrollbar object will pass control must be installed by the application after the scroll object is created, with a call of the composition:

```
xvw_add_callback(scroll_object, XVW_SCROLL_INCR_MOTION,
                 callback, client_data);
```

-and/or -

```
xvw_add_callback(scroll_object, XVW_SCROLL_CONT_MOTION,
                 callback, client_data);
```

The callback routine must be defined according to the syntax:

```
void callback(
   xvobject object,
   kaddr  client_data,
   kaddr  call_data)
{
        int *scrollbar_movement;

        /* see "USING CALLBACKS" for use of client_data */
        /* cast the pointer to the call_data */

        scrollbar_movement = (int) call_data;

     /* now, scrollbar_movement indicates movement of scrollbar thumb */
        :
        :
}
```

As implied in the above specification, the user's movement of the scrollbar thumb causes the *xvwidgets* library to pass software control to the callback routine defined. The callback routine will be called with the call_data containing an integer value representing the movement of the scrollbar thumb from its previous position. A positive movement value indicates that the scrollbar was moved to the left; a negative value indicates that the scrollbar was moved to the right.

N.4. Scrollbar Example

```
#include <design.h>


/*
 *   This example program puts 5 vertical scrollbars & 5 horizontal
 *   scrollbars.  It adds callbacks to the scrollbars to print the value
 *   currently indicated by the scrollbar.  This example also serves to
 *   illustrate the use of call_data.
 */
#define SCROLLBAR_NUM 5

void scrollbar_cb PROTO((xvobject, kaddr, kaddr));

void main(
   int  argc,
   char *argv[])
{
     int      i;
     char     name[KLENGTH];
     xvobject manager;
     xvobject sb;
     xvobject horiz_offset;
     xvobject vert_offset;
```

```
   /* initialize VisiQuest program */
   khoros_initialize(argc, argv, "DESIGN");

/* initialize the xvwidgets library */
if (!xvw_initialize(XVW_MENUS_XVFORMS))
{
   kerror("example", "main", "Cannot open display");
   kexit(KEXIT_FAILURE);
}

/* every GUI has a manager backplane */
manager = xvw_create_manager(NULL, "back");

horiz_offset = NULL;

/*
 *  five vertical scrollbars in a row
 */
for (i = 0; i < SCROLLBAR_NUM; i++)
{
   sprintf(name, "vert_sb_%d", i+1);
   sb  = xvw_create_scrollbar(manager, name);
   xvw_set_attributes(sb,
      XVW_RIGHT_OF,     horiz_offset,    /* to R of neighbor        */
      XVW_ORIENTATION,  KSCROLLBAR_ORIENT_VERT, /* vertical        */
      XVW_CHAR_WIDTH,   2.0,             /* 2 chars wide           */
      XVW_CHAR_HEIGHT,  10.0,            /* 10 chars  high         */
      XVW_SCROLL_MIN,   0.0,             /* min value of 0         */
      XVW_SCROLL_MAX,   100.0,           /* max value of 100       */
      XVW_SCROLL_VALUE, ((double)i)*10.0, /* value = last val * 10  */
      NULL);

   horiz_offset = sb;
   xvw_add_callback(sb, XVW_SCROLL_CONT_MOTION,
               scrollbar_cb, NULL);
   xvw_add_callback(sb, XVW_SCROLL_INCR_MOTION,
                       scrollbar_cb, NULL);
}


/*
 *  five horizontal scrollbars in a row
 */
vert_offset = horiz_offset;
for (i = 0; i < SCROLLBAR_NUM; i++)
{
   sprintf(name, "horiz_sb_%d", i+1);
   sb  = xvw_create_scrollbar(manager, name);
   xvw_set_attributes(sb,
      XVW_BELOW,        vert_offset,      /* under vertical sb set  */
      XVW_ORIENTATION,  KSCROLLBAR_ORIENT_HORIZ, /* horizontal      */
      XVW_CHAR_WIDTH,   20.0,            /* 20 chars wide          */
      XVW_CHAR_HEIGHT,  1.0,             /* 1 chars  high          */
      XVW_SCROLL_MIN,   0.0,             /* min value of 0         */
      XVW_SCROLL_MAX,   100.0,           /* max value of 100       */
      XVW_SCROLL_VALUE, ((double)i)*10.0, /* value = last val * 10  */
      NULL);

   vert_offset = sb;
```

```
        xvw_add_callback(sb, XVW_SCROLL_CONT_MOTION,
                    scrollbar_cb, NULL);
        xvw_add_callback(sb, XVW_SCROLL_INCR_MOTION,
                          scrollbar_cb, NULL);
     }

     /* display & run the program */
     xvf_run_form();
}

void scrollbar_cb (
    xvobject object,
    kaddr    client_data,
    kaddr    call_data)
{
     double value;
     double *movement = (double *) call_data;

        /*
         * a scrollbar object is one of the few objects that uses a call_data
         * structure.  it needs to => return information to the caller,
         * so that information is returned via the call_data.  The data type
         * of the call_data is determined by the object itself;  a scrollbar
         * object uses a call_data of type "double *".  The movement of the
         * scrollbar returned by the call_data is identical to the value
         * obtained by using the XVW_SCROLL_VALUE attribute, so you can
         * get the value of the scrollbar either way.
         */

     xvw_get_attribute(object, XVW_SCROLL_VALUE, &value);
     kfprintf(kstderr, "value = %f\n", value);
     kfprintf(kstderr, "movement = %f\n", *movement);
}
```

O. The Text Widget

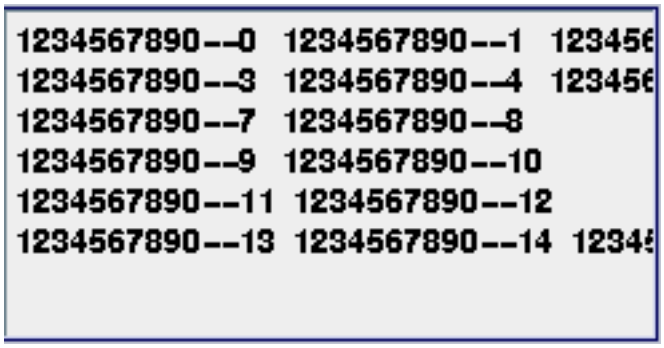


Figure 8: The text widget is most commonly used as a sub-part of other widgets in VisiQuest.

O.1. xvw_create_text() — *create a text object*

Synopsis

```
xvobject xvw_create_text(

    xvobject parent,
    char   *name)
```

Input Arguments

`parent`
: parent of the text object

`name`
: a name for this particular instance of the text object (for use in app-defaults files, etc)

Returns

The text GUI object on success, NULL on failure.

Description

The *text* object allows the display and editing of text.

O.2. About the Text Object

The VisiQuest 2001 *text object* may be single-line or multi-line. It may display read-only text, or allow the editing of read-write text. The source for default text may be specified directly from within the application (hard-coded) or may be contained in a file. The text object supports normal cut and paste operations. It offers a number of key bindings. Word and line wrapping are also supported.

O.2.1. Single-Line vs. Multi-line Text Objects

Text objects may be used to edit and/or display a single line of text or multiple lines of text. Functionality of multi-line text objects is the same as that of single-line text objects, except for the following:

- If text containing a line feed is pasted to a multi-line text object, the line feed is retained; however, in a single-line text object, a line feed is replaced with a space (both types of text objects replace unprintable characters with a space).

- With a single-line text widget, the application may install a callback that will be fired when the user enters <return>. By definition, no such callback may be installed on a multi-line text widget, as <return> in the text of a multi-line text widget must be used to indicate line feed.

- Multi-line text objects may be of any height; single-line text objects will always be 1 character in height.

- Multi-line text objects may do text wrapping if directed to do so by the application; by definition, single-line text objects do not allow text wrapping.

O.2.2. Cursor Placement

Location of the cursor may be set by clicking the left mouse button at the desired location in the text, or by using the supported key bindings (see following section).

O.2.3. Navigation of Multi-line Text Objects

Multi-line text objects may contain more text than can be displayed in the text window. Because automatic addition of scrollbars is not yet supported (but planned for a future VisiQuest release), key bindings must be used to navigate the text. The arrow keys may be used to move up, down, left, and right. Other key bindings are also provided to move to the beginning or the end of a line, or to move up or down a page. Other key bindings allow movement forward or backward word by word, or paragraph by paragraph.

O.2.4. Read-Only vs. Read-Write Text Objects

Text objects may display text that is not writable, or may allow the user to edit the text. Regardless of whether the text is read-only or read-write, a cursor will appear. This is so that the user may use the text object's key bindings to scroll through text that may not all fit within the text window, since automatic addition of vertical and horizontal scrollbars to the text object is not yet supported. Any attempt to edit text in a read-only text

object will ring the bell.

O.2.5. Text Focus

When the pointer is moved into the text window, the border of the text object will be highlighted, and the cursor will blink. The contents of read-write text objects may be modified at this time; the contents of both read-only and read-write text objects may be navigated using key bindings. Clicking in the text window, however, will not cause the text object to grab the keyboard; in other words, if the pointer leaves the text window, text focus will be lost.

O.2.6. Specified Text Source vs. File Containing Text Source

The text object will allow the application to specify the text to be displayed directly (see `XVW_TEXT_STRING`) or may specify the path to a file that contains the text to be displayed (see `XVW_TEXT_FILENAME`). Specify one or the other, not both. When using a file, full paths specified with ${*TOOLBOX*} are recommended.

O.2.7. Text Wrapping

When the text lines of a multi-line text object exceed the width of the text window, *text wrapping* may be implemented. Text wrapping dictates that text lines are to be automatically broken in an attempt to get all lines within the width of the text window. Since single-line text objects only allow a single line of text, text wrapping applies to multi-line text objects only. For multi-line text, the text object supports three wrapping modes:

no wrap
: No text wrapping is done.

word wrap
: The text object will wrap the text at the last word that will fit within the width of the text window. If a single word is wider than the text object, that word will still extend past the right hand side of the window; the user will be required to use key bindings to see the end of the word.

line wrapping
: The text object will wrap text at the last character that will fit within the width of the text window.

O.2.8. Cut and Paste

The text object provides for cut and paste operations.

Text highlighted in another text object, another application, or another window may be pasted into the text object *at the location of the pointer* by pressing the middle mouse button with the pointer at the desired location in the text. To paste text at the location of the cursor, first highlight the desired text. Move the cursor to the desired location, and then press "<Ctrl>+Y" or "<Ctrl>+Insert" to paste the text at the cursor location.

Note that when the cursor is moved, or any key bindings are used to navigate the text window, any portion of the text that is highlighted in the text object will be un-highlighted. However, the previously highlighted text

will remain in the cut buffer until the contents of the cut buffer is over-ridden with a new highlighting operation. Thus, a paste operation will always paste the most recent contents of the cut buffer. If those contents originated in the text object, the source text may or may not be highlighted in the text object, depending on whether any text window navigation was done in the interim.

Highlighting of text to be be stored in the cut buffer may be done in one of four ways:

- An arbitrary text region may be highlighted by pressing the left mouse button at the beginning of the desired text, and holding the mouse button down while moving the mouse to the end of the text.
- A single word may be highlighted by double-clicking the left mouse button on the desired word.
- A line of text may be highlighted by triple-clicking the left mouse button on the desired line.
- The entire contents of the text widget may be highlighted by quadruple-clicking the left mouse button anywhere in the text window.

O.2.9. Text Object Key Bindings

The text object supports a variety of key bindings. These are as described in the following table. Many are standard Motif-compliant key bindings; others are provided to emulate the most basic of the emacs key bindings.

Text Object Key Bindings

Key Binding	Description
Up arrow {Ctrl} p	Moves cursor one line up
Down arrow {Ctrl} n	Moves cursor one line down
Left arrow {Ctrl} b	Moves cursor one character to the left (if cursor has to move up a line, x location will be reset to 0)
Right arrow {Ctrl} f	Moves cursor one character to the right (if cursor has to move down a line, x location will be reset to 0)
Home, {Ctrl} a	Moves to beginning of current line
End, {Ctrl} e	Moves to end of current line
Page Up Prior {Meta} v	Moves back (up) one page
Page Down Next {Ctrl} v	Moves forward (down) one page

Text Object Key Bindings	
Key Binding	**Description**
{Ctrl} Up Arrow	Moves to beginning of current paragraph (or to previous paragraph if cursor already at paragraph beginning)
{Ctrl} Down Arrow	Moves to beginning of next paragraph
{Ctrl} Left Arrow {Meta} b	Moves to beginning of current word (or to beginning of next word if cursor is already at a word beginning)
{Ctrl} Right Arrow {Meta} f	Moves to beginning of next word
{Ctrl} Home	Moves to beginning of text
{Ctrl} End	Moves to end of text
{Ctrl} Insert, {Ctrl} y	Pastes contents of cut buffer to location of cursor

O.2.10. Attributes of the Text Object

Summary of Text Attributes	
Attribute	**Description**
XVW_TEXT_EDIT_TYPE	The text in the text object may be read-only, or read-write. Specify KTEXT_READ for the former, or KTEXT_EDIT
XVW_TEXT_ENTER_STRING	If desired, *xvw_add_callback()* may be used to install a callback on the text object text object which will be fired when the user enters <return> in the text window. must be reserved to indicate a carriage return. The callback should be installed as follows: `xvw_add_callback(text_object, XVW_TEXT_ENTER_STRING, callback, client_data);` The string that was entered will be sent to the callback as the call_data. Before using, the string must be cast to its proper data type, as in: kstring string = (kstring) call_data;
XVW_TEXT_FILENAME	The name of the file whose contents is displayed in the text object. It is mutually exclusive with XVW_TEXT_STRING; use one or the other, not both simultaneously.
XVW_TEXT_INSERT_POS	The location of the cursor in the string, where each number represents a character. For example, if XVW_TEXT_INSERT_POS was set to 5, that would mean that the cursor appeared 5 letters into the string.
XVW_TEXT_LEFT_MARGIN	The number of pixels from the leftmost edge of the text object that the string will be displayed.
XVW_TEXT_MULTILINE	Control whether or not more than one line of text can be displayed. Set to TRUE if you want to display more than one line of text.

Summary of Text Attributes	
Attribute	**Description**
XVW_TEXT_STRING	The string that is displayed in the text object. It is mutually exclusive with XVW_TEXT_FILENAME; use one or the other, not both simultaneously.
XVW_TEXT_WRAP	Used with multi-line text objects only, this attribute controls if and how text is wrapped when it is longer than the text object's width. The text may be left unwrapped, it may wrap on a line which may divide words, or attempt to wrap on words which will leave words intact. KTEXT_WRAP_NEVER- don't wrap text KTEXT_WRAP_LINE- wrap text on lines (may break up words) KTEXT_WRAP_WORD- wrap text on words (attempt to not break up words). With single-line text objects, this attribute is ignored; wrapping is never done.

Descriptions of Text Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_TEXT_EDIT_TYPE (N/A)	int	KTEXT_EDIT	KTEXT_READ KTEXT_EDIT
XVW_TEXT_ENTER_STRING (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_TEXT_FILENAME (N/A)	char *	NULL	name of any existing file with readable text
XVW_TEXT_INSERT_POS (TextLabel.labelCursorPos)	int	0	0 <= value <= length of currently displayed text
XVW_TEXT_LEFT_MARGIN (textLeftMargin)	unsigned short	0	values >= 0
XVW_TEXT_MULTILINE (textMultiline)	int	FALSE	TRUE/FALSE
XVW_TEXT_STRING (N/A)	char *	NULL	any printable text
XVW_TEXT_WRAP (textWrap)	int	KTEXT_WRAP_NEVER	KTEXT_WRAP_WORD KTEXT_WRAP_NEVER KTEXT_WRAP_LINE

O.3. About Callbacks on Text Objects

When the user enters <return> in a single-line text widget, software control may be passed to an installed callback. The callback to which the text object will pass control must be installed on the text object by the application after the text object is created, with a call of the composition:

```
xvw_add_callback(text_object, XVW_TEXT_ENTER_STRING,
                 callback, client_data);
```

The callback routine must be defined according to the syntax:

```
void callback(
   xvobject object,
   kaddr  client_data,
   kaddr  call_data)
{
     /* cast call_data so that "contents" is the text currently
        appearing in the text object */
     kstring contents = *(kstring *) call_data;

     /* do something with the "contents" string */
}
```

P. The Viewport Object

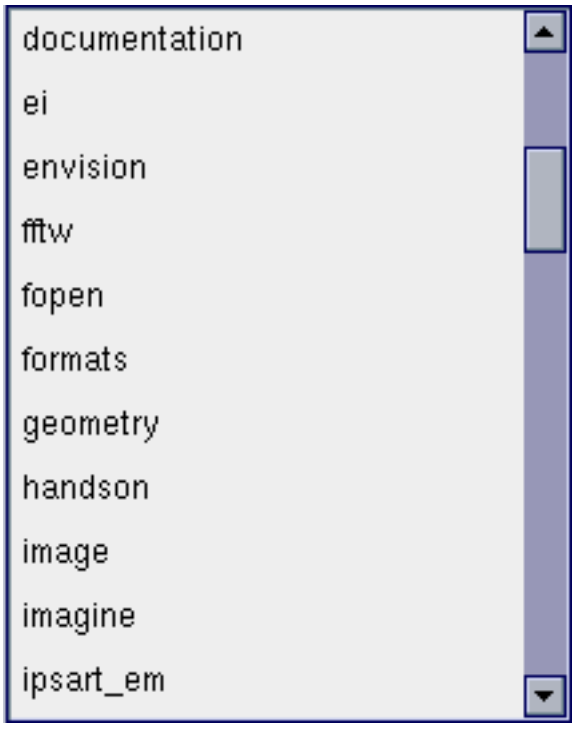


Figure 9: The Viewport GUI object provides a scrollable area in which various other GUI objects may be placed. Here, the compound list GUI object uses a viewport as one of its two components.

P.1. xvw_create_viewport() — *create a viewport object*

Synopsis

```
xvobject xvw_create_viewport(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: parent of the viewport object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The viewport object on success, NULL on failure

Description

Creates a viewport widget. The *viewport* provides a way of restricting a potentially overlarge display to a predefined area, while at the same time allowing its contents (consisting of other GUI objects) to occupy as much space as necessary.

The viewport object is made up of four other GUI objects: a "plane" object, a clip object, a vertical scrollbar, and a horizontal scrollbar.

The plane object component of the viewport object implements a "virtual area", only part of which can be seen in the viewport. The plane object is the actual parent of the viewport's children; in this way, the the object(s) created inside the viewport can take up more space than the viewport itself.

The clip object component of the viewport is what dictates the size of the viewport itself. By clipping out all portions of the plane object that lie outside the bounds of the viewport size, the contents of the viewport are only visible when they are in the predefined range.

The viewport contains horizontal and vertical scrollbars; these scrollbars are used to control which portion of the plane object is visible in the viewport.

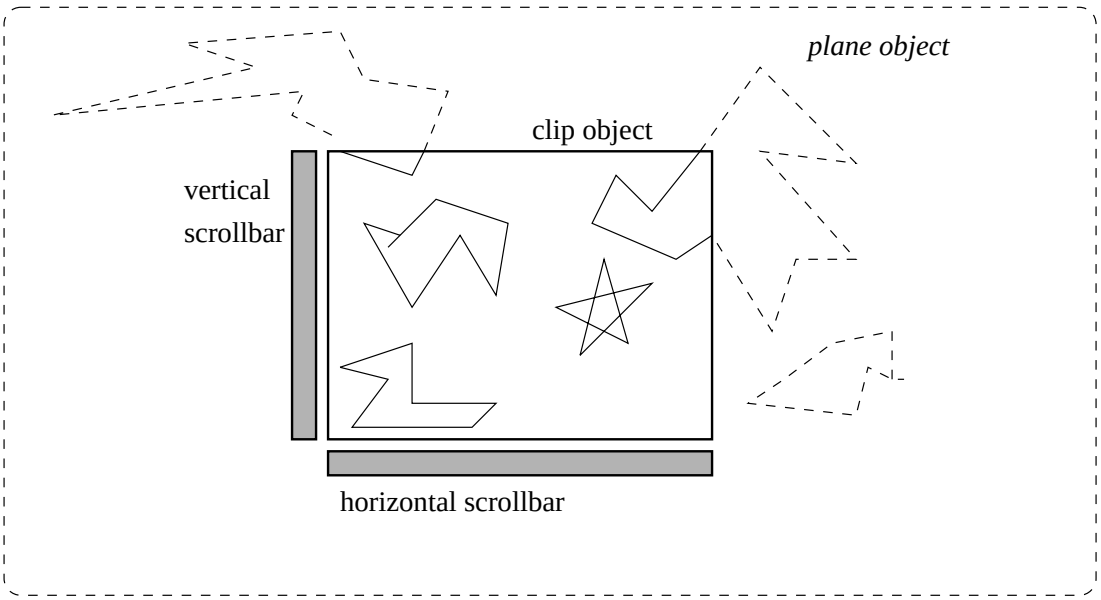


Figure 10: The viewport object is made up of a "plane" object, a clip object, a vertical scrollbar, and a horizontal scrollbar.

P.2. Attributes of the Viewport Object

Summary of Viewport Attributes	
Attribute	**Description**
XVW_VP_ALLOW_HORIZ	Set to TRUE if a horizontal scrollbar should appear if it is needed.
XVW_VP_ALLOW_VERT	Set to TRUE if a vertical scrollbar should appear if it is needed.
XVW_VP_CLIP_OBJECT	This is the clip object component of the viewport; it serves as the viewport to the plane object.
XVW_VP_FORCE_HORIZ	Set to TRUE if a horizontal scrollbar should always appear. Only valid if XVW_VP_ALLOW_HORIZ is TRUE.
XVW_VP_FORCE_VERT	Set to TRUE if a vertical scrollbar should always appear. Only valid if XVW_VP_ALLOW_VERT is TRUE.
XVW_VP_HORIZONTAL_SCROLLBAR	This is a read-only attribute that will return the pointer to the xvobject for the horizontal scrollbar.
XVW_VP_PLANE_OBJECT	This is the plane object component of the viewport; children of the viewport created by the application are actually created as children of this plane object, *not* of the viewport itself.
XVW_VP_USE_BOTTOM	Set to TRUE if the horizontal scrollbar should appear at the bottom of the viewport; set to FALSE to make the horizontal scrollbar appear at the top of the viewport.
XVW_VP_USE_RIGHT	Set to TRUE if the vertical scrollbar should appear at the right of the viewport; set to FALSE to make the vertical scrollbar appear at the left of the viewport.
XVW_VP_VERTICAL_SCROLLBAR	This is a read-only attribute that will return the pointer to the xvobject for the vertical scrollbar.
XVW_VP_XOFFSET	The offset of the viewport in pixels from the left side.
XVW_VP_YOFFSET	The offset of the viewport in pixels from the top.

Descriptions of Viewport Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_VP_ALLOW_HORIZ (vpAllowHoriz)	int	FALSE	TRUE/FALSE
XVW_VP_ALLOW_VERT (vpAllowVert)	int	TRUE	TRUE/FALSE
XVW_VP_CLIP_OBJECT (N/A)	xvobject	NULL	valid xvobject
XVW_VP_FORCE_HORIZ (vpForceHoriz)	int	FALSE	TRUE/FALSE
XVW_VP_FORCE_VERT (vpForceVert)	int	TRUE	TRUE/FALSE

Descriptions of Viewport Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_VP_HORIZONTAL_SCROLLBAR (N/A)	xvobject	NULL	valid xvobject
XVW_VP_PLANE_OBJECT (N/A)	xvobject	NULL	valid xvobject
XVW_VP_USE_BOTTOM (vpUseBottom)	int	TRUE	TRUE/FALSE
XVW_VP_USE_RIGHT (vpUseRight)	int	FALSE	TRUE/FALSE
XVW_VP_VERTICAL_SCROLLBAR (N/A)	xvobject	NULL	valid xvobject
XVW_VP_XOFFSET (vpXoffset)	int	0	value >= 0
XVW_VP_YOFFSET (vpYoffset)	int	0	values >= 0

P.3. Complete Resource Set of the Viewport Object

The complete resource set for the viewport object includes:

1. The viewport object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Section C.2, "Attributes of the VisiQuest 2001 Manager Object".

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

P.4. Example using the Viewport Object

An example program using the textinput object can be found in `$DESIGN/examples/xvwidgets/viewport/example.c`.

This page left intentionally blank

Table of Contents

Chapter 3

Xvobjects

Chapter 3 - Xvobjects

A. Introduction

The *xvobjects* library provides a number of special-purpose GUI objects that were written especially for use in VisiQuest 2001. Compound GUI objects for input of parameters are provided, such as the browser object, the double object, the float object, the integer object, and the textinput object. In addition, the *xvobjects* library contains the GUI objects intended for interactive notification, specifically the error object, the info object, the warn object, and the notifywindow object. GUI objects meant for use as special-purpose backplanes are also included here, such as the canvas, the layout, and the rootwindow objects.

Available Functions

- *xvw_create_browser()* - create a browser GUI object
- *xvw_create_canvas()* - create a canvas object
- *xvw_create_connection()* - create a connection object
- *xvw_create_double()* - creates a double object
- *xvw_create_error()* - create an error object
- *xvw_create_float()* - create a float object
- *xvw_create_help()* - create a help object
- *xvw_create_info()* - create an info object
- *xvw_create_inputfile()* - create a inputfile GUI object
- *xvw_create_integer()* - create an integer GUI object
- *xvw_create_layout()* - create a layout object
- *xvw_create_notifywindow()* - create a notifywindow object
- *xvw_create_outputfile()* - create a outputfile GUI object
- *xvw_create_rootwindow()* - create a rootwindow object
- *xvw_create_textdisplay()* - create a textdisplay object
- *xvw_create_textinput()* - create a textinput object
- *xvw_create_warn()* - create a warning object

B. The Browser Object

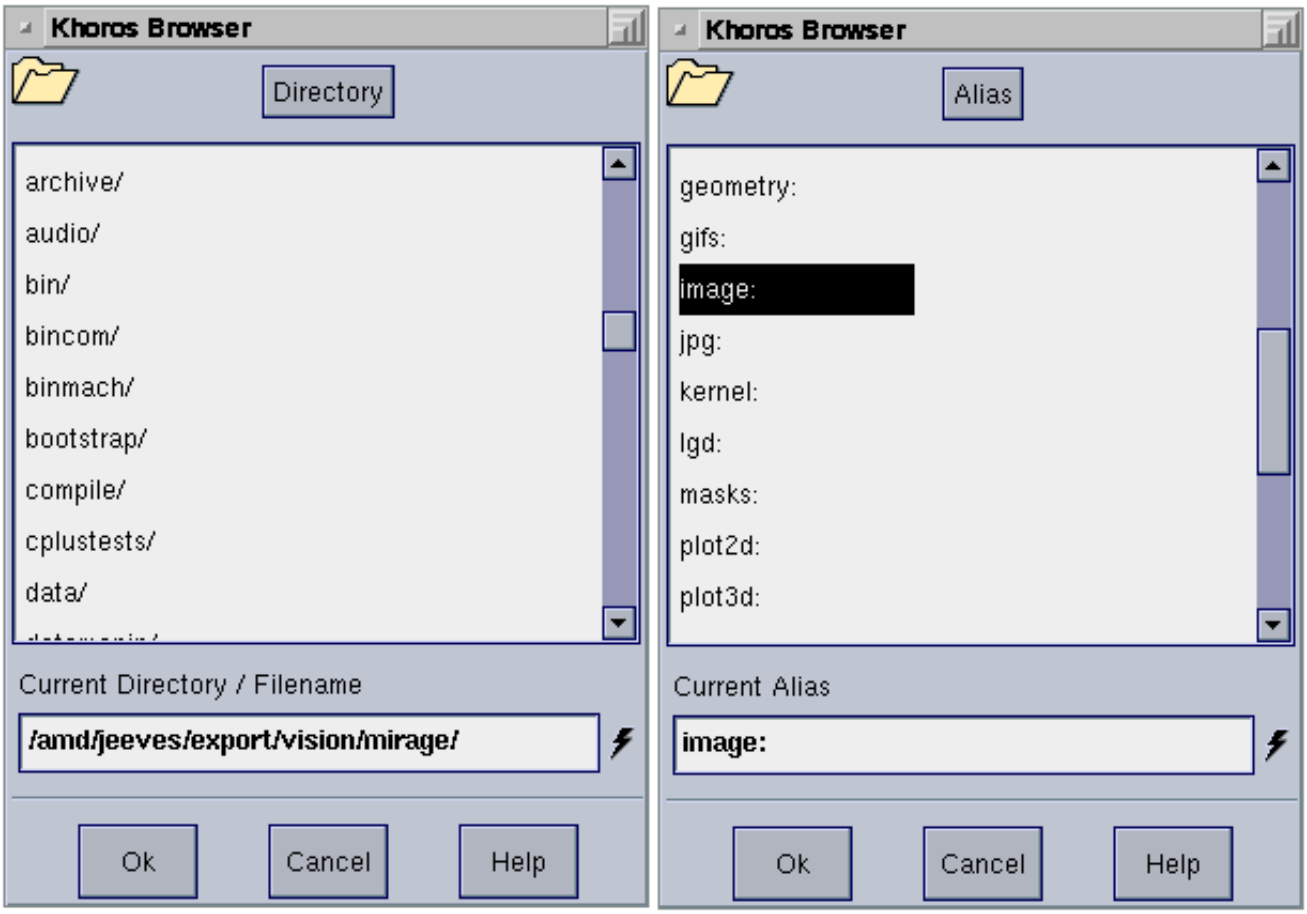


Figure 1: The browser object is used throughout the VisiQuest system; it is popped up when the user clicks on the label button of an InputFile or OutputFile selection.

B.1. xvw_create_browser() — *create a browser GUI object*

Synopsis

```
xvobject xvw_create_browser(
    xvobject parent,
    char     *name)
```

Input Arguments

`parent`
: parent of the browser object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The browser object on success, NULL on failure

Description

The browser GUI object allows the user to enter an input file. It features a list object in which the user may select a file, or a text box in which the user may type in the filename; the filename is registered when the user hits <cr> or selects a file from the browser list.

The browser GUI object consists of a manager object with six children: two button object, a list object, a label object, a textinput object, and a textinput object.

A callback can be installed on the browser object, which will be fired when the user enters a new filename.

B.2. Attributes of the Browser Object

Summary of Browser Attributes

Attribute	Description
XVW_BROWSER_ALIASES_PIXMAP	This is the pixmap that appears to the upper left of the browser object when the browser is in aliases mode. Candidates for the value of this attribute may be created with the use of *XCreatePixmap()*; see *The Xlib Reference Manual* by O'Reilly and Associates. Note that this attribute is mutually exclusive with XVW_BROWSER_PIXMAPFILE; specify one or the other, not both.
XVW_BROWSER_ALIASES_PIXMAPFILE	This is the file defining the pixmap that appears at the upper left of the browser object when the browser is in aliases mode.

XVW_BROWSER_CALLBACK	If desired, *xvw_add_callback()* may be used to install a callback on the browser object which will be fired when the user enters new text and presses <cr>. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(browserobj, XVW_BROWSER_CALLBACK,`
	`browser_cb, client_data);`
	Note that the current text appearing in the browser object will be passed to the callback in the *call_data*. The value must be cast to a string before use, as in:
	`char *string = *((char **) call_data);`
	Alternatively, the filename may be obtained with XVW_BROWSER_TEXT.

XVW_BROWSER_DESTROY_ON_QUIT	If FALSE, the browser object will not be destroyed when the user clicks on the "Cancel" button or selects a file, but will simply be unmapped so that it can be used again.

Summary of Browser Attributes	
Attribute	**Description**
XVW_BROWSER_DIRECTORY	This attribute allows you to set or get the directory currently selected within the browser.
XVW_BROWSER_DIRECTORY_PIXMAP	This is the pixmap that appears to the upper left of the browser object when the browser is in directory mode. Candidates for the value of this attribute may be created with the use of *XCreatePixmap()*; see *The Xlib Reference Manual* by O'Reilly and Associates. Note that this attribute is mutually exclusive with XVW_BROWSER_PIXMAPFILE; specify one or the other, not both.
XVW_BROWSER_DIRECTORY_PIXMAPFILE	This is the file defining the pixmap that appears at the upper left of the browser object when the browser is in directory mode.
XVW_BROWSER_TYPE	This attribute allows you to set the type of browser, which modifies the browser behavior.

Descriptions of Browser Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_BROWSER_ALIASES_PIXMAP (N/A)	Pixmap	The "aliases browser" pixmap.	Valid Pixmap structure
XVW_BROWSER_ALIASES_PIXMAPFILE (browserAliasesPixmapfile)	char *	The browser.xpm file in the *xvobjects/misc/pixmaps* directory, which defines the "aliases browser" pixmap.	The full path to a valid xpm or xbm file, defining the desired pixmap. Note that the path may contain $TOOLBOX.
XVW_BROWSER_CALLBACK (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_BROWSER_DESTROY_ON_QUIT (browserDestroyOnQuit)	int	TRUE	TRUE/FALSE
XVW_BROWSER_DIRECTORY (browserDirectory)	char *	NULL	any legal directory path
XVW_BROWSER_DIRECTORY_PIXMAP (N/A)	Pixmap	The "directory browser" pixmap.	Valid Pixmap structure

Descriptions of Browser Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_BROWSER_DIRECTORY_PIXMAPFILE (browserDirectoryPixmapfile)	char *	The browser.xpm file in the *xvobjects/misc/pixmaps* directory, which defines the "directory browser" pixmap.	The full path to a valid xpm or xbm file, defining the desired pixmap. Note that the path may contain $TOOLBOX.
XVW_BROWSER_TYPE (browserDirectory)	char *	NULL	any legal directory path

B.3. Complete Resource Set of the Browser Manager Object

The complete resource set for the browser object includes:

1. The browser object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3,"The kwidgets Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object".

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

B.4. Example using the Browser Object

An example program using the browser object can be found in `$DESIGN/examples/xvobjects/browser/example.c`.

```
#include <design.h>

/*
 *  This example creates a browser object and installs on it a callback
 *  which will print the file that was selected by the user.
 */
static void browser_cb PROTO((xvobject, kaddr, kaddr));

main(
   int  argc,
   char *argv[])
{
     xvobject browser;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidgets lib */
```

```
        if (!xvw_initialize(XVW_MENUS_XVFORMS))
        {
           kerror(NULL, "main", "unable to open display");
           kexit(KEXIT_FAILURE);
        }

        browser = xvw_create_browser(NULL, "browser");
        xvw_add_callback(browser, XVW_BROWSER_CALLBACK, browser_cb, NULL);

        /* that's all! */
        xvf_run_form();
    }

    static void browser_cb(
        xvobject object,
        kaddr    client_data,
        kaddr    call_data)
    {
        char *filename = *((char **) call_data);

        if (filename == NULL)
           kinfo(KFORCE, "No filename selected");
        else kinfo(KFORCE, "The filename selected was '%s'", filename);
    }
```

C. The Canvas Object

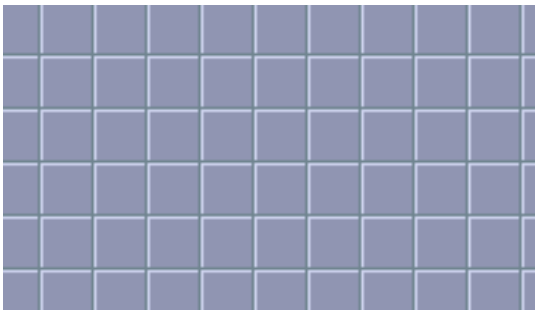


Figure 2: The Canvas GUI object provides an area in which other GUI objects may be created.

C.1. xvw_create_canvas() — *create a canvas object*

Synopsis

```
xvobject xvw_create_canvas(
    xvobject parent,
    char     *name)
```

Input Arguments

`parent`
: parent of the canvas object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The canvas object on success, NULL on failure

Description

The canvas object is essentially a manager object combined with a virtual window size. The virtual window is a viewport where a canvas may contain a vertical scrollbar, a horizontal scrollbar, or both. In general, it is used to confine large visual displays to a predefined area. The canvas object may have one or many children; usually, the object(s) created inside the canvas will take up more space than the canvas itself, so that the scrollbar(s) of the canvas may be used to view the entire display. The canvas object is similar to the viewport object, except the canvas provides an area for displaying graphical objects, while the viewport object is typically only used to contain GUI objects. The canvas object allows the application to set a default width and height, while the viewport will shrink its area to the minimum size required to lay out all its children.

The canvas object also offers a global clipboard, which acts as a common cut and paste area for use by all canvas objects that may be displayed at the same time, providing an easy method for the user to

transfer objects from one canvas to another.

C.2. Attributes of the Canvas Manager Object

Summary of Canvas Attributes	
Attribute	**Description**
XVW_CANVAS_COPY	This action attribute *copies* the currently selected objects in the cavas into the global clipboard. The clipboard is global to all displayed canvas objects, regardless of whether or not the different canvas objects have different parents. If no object on the canvas has been selected, a warning message will be issued.
XVW_CANVAS_CUT	This action attribute *moves* the currently selected objects in the canvas into the global clipboard. The clipboard is global to all displayed canvas objects, regardless of whether or not the different canvas objects have different parents. If no object on the canvas has been selected, a warning message will be issued.
XVW_CANVAS_DELETE	This action attribute deletes all currently selected objects from the canvas. The objects are temporarily saved (until the next time the XVW_CANVAS_DELETE action attribute is used), so that the delete operation can be undone by setting the XVW_CANVAS_UNDO action attribute. If no object on the canvas has been selected, a warning message will be issued.
XVW_CANVAS_DUPLICATE	This action attribute duplicates the currently selected objects in the canvas. Any selected object duplicated; the duplicated objects are then selected and positioned 10 pixels below and to the right of the originals. If no object on the canvas has been selected, a warning message will be issued.
XVW_CANVAS_GRID	This attribute indicates when a grid should be displayed on the canvas. KMANAGER_GRID_OFF specifies that the grid is always off. KMANAGER_GRID_ON specifies that the grid is always on. KMANAGER_GRID_EDIT specifies that the grid is only on when the canvas is in "edit mode". The canvas can be put in edit mode by the user, or through the application by setting the manager object attribute XVW_EDIT_MODE_ON to TRUE. KMANAGER_GRID_SELECT specifies that the grid is only displayed when a child of the canvas has been selected by the user, or through the application by setting the manager object attribute XVW_SELECT_ADD to TRUE.
XVW_CANVAS_GRIDSIZE	The dimensions of the grid, given in pixels.
XVW_CANVAS_HEIGHT	The canvas height of the canvas workspace plane.
XVW_CANVAS_LOWER	This action attribute lowers the currently selected objects in the canvas beneath any other objects that may overlap the same space. If no object on the canvas has been selected, a warning message will be issued.

Summary of Canvas Attributes	
Attribute	**Description**
XVW_CANVAS_PASTE	This action attribute pastes the contents of the currently selected objects in the global clipboard into the canvas object. The clipboard is global to all displayed canvas objects, regardless of whether or not the different canvas objects have different parents. If no object on the clipboard has been selected, a warning message will be issued.
XVW_CANVAS_RAISE	This action attribute raises the currently selected objects in the canvas beneath any other objects that may overlap the same space. If no object on the canvas has been selected, a warning message will be issued.
XVW_CANVAS_SELECTALL	This action attribute automatically selects all objects in the canvas, unless the objects have been designated as "unselectable". An object is made "unselectable" if its XVW_SELECTABLE attribute has been set to FALSE.
XVW_CANVAS_UNDO	This action attribute restores the object(s) that were most recently deleted using the XVW_CANVAS_DELETE action attribute. If no object on the canvas has been selected, a warning message will be issued.
XVW_CANVAS_UNSELECTALL	This action attribute automatically unselects all currently selected objects.
XVW_CANVAS_WIDTH	The canvas width of the canvas workspace plane.

Descriptions of Canvas Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_CANVAS_COPY (N/A)	int	N/A	TRUE
XVW_CANVAS_CUT (N/A)	int	N/A	TRUE
XVW_CANVAS_DELETE (N/A)	int	N/A	TRUE
XVW_CANVAS_DUPLICATE (N/A)	int	N/A	TRUE
XVW_CANVAS_GRID (canvasGrid)	int	KMANAGER_GRID_ON	KMANAGER_GRID_OFF KMANAGER_GRID_ON KMANAGER_GRID_EDIT KMANAGER_GRID_SELECT
XVW_CANVAS_GRIDSIZE (canvasGridsize)	int	15	value > 0
XVW_CANVAS_HEIGHT (N/A)	int	3000	value > 0
XVW_CANVAS_LOWER (N/A)	int	N/A	TRUE
XVW_CANVAS_PASTE (N/A)	int	N/A	TRUE

Descriptions of Canvas Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_CANVAS_RAISE (N/A)	int	N/A	TRUE
XVW_CANVAS_SELECTALL (N/A)	int	N/A	TRUE
XVW_CANVAS_UNDO (N/A)	int	N/A	TRUE
XVW_CANVAS_UNSELECTALL (N/A)	int	N/A	TRUE
XVW_CANVAS_WIDTH (N/A)	int	3000	value > 0

C.3. Complete Resource Set of the Canvas Manager Object

The complete resource set for the canvas object includes:

1. The canvas object attribute resource set, given in the previous section.

2. The viewport object attribute resource set, given in Section U.2, "Attributes of the Viewport Object".

3. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3,"The kwidgets Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object".

4. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

C.4. Example using the Canvas Object

An example program using the canvas object can be found in `$DESIGN/examples/xvobjects/canvas/example.c`.

```
#include <design.h>

/*
 *  This example creates a parent object with a button and a canvas object;
 *  then creates a button label on the canvas to act as a sample object
 *  that can be selected and moved about.
 *
 *  The button above the canvas may be used to change
 *  the background pixmap of the canvas. (Mostly to show off all the
 *  cool background pixmaps available).
 */
static void change_background PROTO((xvobject, kaddr, kaddr));
```

```
main(
   int  argc,
   char *argv[])
{
     int gridsize = 20;
     xvobject parent, canvas, button;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /* create parent to act as backplane */
     parent = xvw_create_manager(NULL, "parent");

     /*
      *  put a button at the top;
         *  install callback so user can change background
      */
     button = xvw_create_button(parent, "button");
     xvw_set_attributes(button,
                  XVW_BELOW,      NULL,
                  XVW_RIGHT_OF,   NULL,
                  XVW_LEFT_OF,    NULL,
                  XVW_LABEL,     "Change background",
                  NULL);

     canvas = xvw_create_canvas(parent, "canvas");
     xvw_set_attributes(canvas,
          XVW_CANVAS_GRIDSIZE, gridsize,
          XVW_BELOW,           button,
          XVW_WIDTH,           512,
          XVW_HEIGHT,          512,
          NULL);

     xvw_add_callback(button, XVW_BUTTON_SELECT, change_background, canvas);

     /* put a button near the top of the canvas to be an object
        that can be moved around */
     button = xvw_create_button(canvas, "button");
     xvw_set_attributes(button,
                    XVW_LABEL,       "sample object",
                    XVW_CHAR_XPOS,   10.0,
                    XVW_CHAR_YPOS,   10.0,
                    XVW_CHAR_HEIGHT, 2.0,
                    NULL);


     /* and we're off! */
     xvf_run_form();
}

static void change_background(
    xvobject object,
```

```
    kaddr    client_data,
    kaddr    call_data)
{
     int      indx;
 static int      count = 0;
     xvobject canvas = (xvobject) client_data;

 static char *filenames[] = {"backgrounds:hex_grid",
                   "backgrounds:brown_marble",
                   "backgrounds:black_marble",
                   "backgrounds:esher_grid",
                   "backgrounds:purple_marble",
                   "backgrounds:gravel_marble",
                   "backgrounds:purple_maze",
                   "backgrounds:wood_pulp",
                   "backgrounds:bw_marble"};

     indx = count % knumber(filenames);
     xvw_set_attribute(canvas, XVW_BACKGROUND_PIXMAPFILE, filenames[indx]);

     count++;
}
```

D. The Connection Object

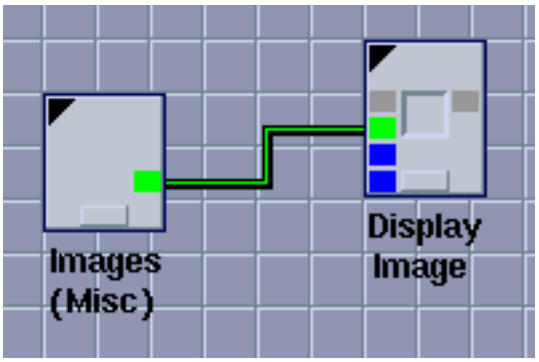


Figure 3: The connection object is used in cantata to connect two glyphs together.

D.1. xvw_create_connection() — *create a connection object*

Synopsis

```
xvobject xvw_create_connection(
   xvobject parent,
   char   *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The connection object on success, NULL on failure.

Description

The connection object provides a mechanism for visually linking two objects. It is used in cantata to connect glyph objects together, but can be used to connect any two objects. A line will be drawn between the two objects; the line will resize and redraw if either of the two objects are moved. Several styles of connections are provided, which can be used to change the appearance of the connection.

D.2. Attributes of the Connection Object

Summary of Connection Attributes	
Attribute	**Description**
XVW_CONNECTION_BEGIN	This specifies the first of the two visual or GUI objects to be connected.
XVW_CONNECTION_END	This specifies the second of the two visual or GUI objects to be connected.
XVW_CONNECTION_LINEWIDTH	Controls the line width of connections
XVW_CONNECTION_TYPE	This specifies the type of connection. May be one of: KCONNECTION_TYPE_LINEAR: connects the objects with straight lines, acute and obtuse angles if necessary. KCONNECTION_TYPE_MANHATTAN: connects the objects with straight lines, squares with right angles when necessary. KCONNECTION_TYPE_SPLINE: connects the objects with a curved line. KCONNECTION_TYPE_HEXAGON: finishes angles with a hexagon. KCONNECTION_TYPE_DIAMOND: finishes angles with a diamond.

Descriptions of Connection Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_CONNECTION_BEGIN (N/A)	xvobject	NULL	valid visual or GUI object
XVW_CONNECTION_END (N/A)	xvobject	NULL	valid visual or GUI object
XVW_CONNECTION_LINEWIDTH (connectionLinewidth)	int	1	1-9
XVW_CONNECTION_TYPE (connectionType)	int	KCONNECTION_TYPE_MANHATTAN	KCONNECTION_TYPE_LINEAR KCONNECTION_TYPE_MANHATTAN KCONNECTION_TYPE_SPLINE KCONNECTION_TYPE_HEXAGON KCONNECTION_TYPE_DIAMOND

D.3. Complete Resource Set of the Connection Object

The complete resource set for the connection object includes:

1. The connection object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3,"The kwidgets Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object".

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

.section 2 "Example using the Connection Object"

An example program using the Connection object can be found in `$DESIGN/examples/xvlang/connection/example.c`. This program is as follows:

E. The Double Object

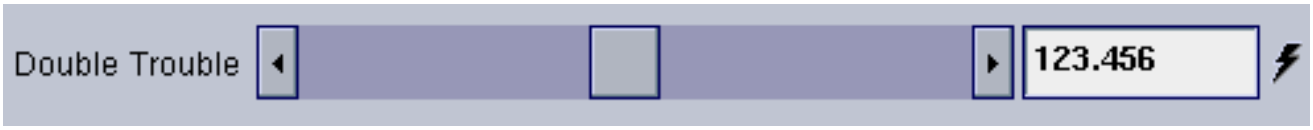


Figure 4: The Double GUI object provides an double text and scrollbar window in which the user may enter an double value.

E.1. xvw_create_double() — *creates a double object*

Synopsis

```
xvobject xvw_create_double(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: parent of the double object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The double GUI object on success, NULL on failure.

Description

The double GUI object allows the user to enter a double precision value. It features a text box in which the user may explicitly enter the double value; the double value is registered when the user hits <cr>. The double value may be bounded by a minimum and maximum value, if desired.

A scroll bar provides an alternate way for the user to specify the double value. The double object consists of a manager object with four children: a label object, a text object, a scrollbar, and a pixmap object.

A callback can be installed on the double object, which will be fired when the user enters a new double value.

E.2. Attributes of the Double Object

Summary of Double Attributes	
Attribute	**Description**

XVW_DOUBLE_CALLBACK	If desired, *xvw_add_callback()* may be used to install a callback on the double object which will be fired when user enters a new double value, either by typing it in and pressing <cr>, or by using the scrollbar. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(doubleobj, XVW_DOUBLE_CALLBACK,`
	`doubleobj_cb, client_data);`
	Note that the current double precision value of the double object will be passed to the callback in the *call_data*. The value must be cast to a double pointer before use, as in:
	`double *value = (double *) call_data;`
	Alternatively, the double value may be obtained with XVW_DOUBLE_VALUE.

XVW_DOUBLE_CRLABEL_OBJECT	This *read-only* attribute allows you to obtain the pixmap object component of the the double object that indicates a "live" selection.
XVW_DOUBLE_INCREMENT	The value to increment the scroll bar thumb.
XVW_DOUBLE_LABEL	This is the text that will appear in the label object component of the double object. Provide text appropriate as a title of the double object.
XVW_DOUBLE_LABEL_OBJECT	This *read-only* attribute allows you to obtain the label object component of the the double object.
XVW_DOUBLE_MAXVALUE	The maximum double value allowed. If both XVW_DOUBLE_MINVALUE and XVW_DOUBLE_MAXVALUE are 0.0, the double value is unbounded.
XVW_DOUBLE_MINVALUE	The minimum double value allowed. If both XVW_DOUBLE_MINVALUE and XVW_DOUBLE_MAXVALUE are 0.0, the double value is unbounded.
XVW_DOUBLE_SCROLLBAR_OBJECT	This *read-only* attribute allows you to obtain the scrollbar object component of the the double object.
XVW_DOUBLE_TEXT_OBJECT	This *read-only* attribute allows you to obtain the text object component of the the double object.
XVW_DOUBLE_VALUE	The double precision value which is currently displayed in the text object. This attribute can be used to initialize the double value to be displayed in the text object, or to acquire a double value that has been entered by the user in the text object.

Descriptions of Double Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_DOUBLE_CALLBACK (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_DOUBLE_CRLABEL_OBJECT (N/A)	xvobject	NULL	The pixmap object (read-only).
XVW_DOUBLE_INCREMENT (N/A)	double	calculated using size of scrollbar	values < (maxvalue - minvalue)/2
XVW_DOUBLE_LABEL (N/A)	char *	NULL	any printable text
XVW_DOUBLE_LABEL_OBJECT (N/A)	xvobject	NULL	The label object (read-only).
XVW_DOUBLE_MAXVALUE (N/A)	double	1.0	any double value
XVW_DOUBLE_MINVALUE (N/A)	double	0.0	any double value
XVW_DOUBLE_SCROLLBAR_OBJECT (N/A)	xvobject	NULL	The scrollbar object (read-only).
XVW_DOUBLE_TEXT_OBJECT (N/A)	xvobject	NULL	The text object (read-only).
XVW_DOUBLE_VALUE (N/A)	double	0.0	any double value

E.3. Complete Resource Set of the Double Object

The complete resource set for the double object includes:

1. The double object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3,"The kwidgets Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object".

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

.section 2 "Example using the Double Object"

An example program using the double object can be found in `$DESIGN/examples/xvobjects/double/example.c`.

```
#include <design.h>

/*
 *  This example creates a simple double GUI object, which may
 *  be used for allowing the user to enter a double.
 *
 *  A callback is installed on the double object so that when the
 *  user changes the value of the double either by entering a number
 *  in the text parameter box and hitting <cr> or by using the scrollbar,
 *  the current value is printed to the tty.
 *
 *  Note that the double object should *not* be created directly in an
 *  xvroutine, as use of the Double (-h) UIS line in the *.form file is
 *  both easier to use and a more standard use of the VisiQuest system.
 *  However, the double object is provided for use with hybrid xvroutines,
 *  (such as this example) which do not use a "formalized" GUI as defined
 *  in a *.form file.
 */
static void double_cb PROTO((xvobject, kaddr, kaddr));

main(
   int  argc,
   char *argv[])
{
     xvobject manager;
     xvobject object;

        /* initialize VisiQuest program
        khoros_initialize(argc, argv, "DESIGN");
*/

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /* create a manager backplane to be a parent for the double object */
     manager = xvw_create_manager(NULL, "parent");
     xvw_set_attributes(manager,
                  XVW_WIDTH,       500,
                     XVW_HEIGHT,      100,
                  NULL);

     /*
         *  Create the double object.  give it a label, a default value,
         *  and bound it with a minimum value of 0 and a maximum value of
         *  255.  tack it horizontally to the parent so that it spans the
         *  width of the manager backplane.  center it in the middle of
         *  the parent.
      */
     object = xvw_create_double(manager, "double");
     xvw_set_attributes(object,
          XVW_DOUBLE_LABEL,    "Double Trouble",
          XVW_DOUBLE_MINVALUE,  0.0,
          XVW_DOUBLE_MAXVALUE,  255.0,
          XVW_DOUBLE_VALUE,     123.456,
          XVW_TACK_EDGE,        KMANAGER_TACK_HORIZ,
```

```
            XVW_ABOVE,      NULL,
            XVW_BELOW,      NULL,
            NULL);

     xvw_add_callback(object, XVW_DOUBLE_CALLBACK, double_cb, NULL);

     /* display & run the program */
     xvf_run_form();
}

/*
 *  the callback for the double will be fired when the user changes the
 *  value of the double, either by using the scrollbar or by entering
 *  a value in the text parameter box & hitting <cr>.  this callback simply
 *  prints the current value of the double.
 */
static void double_cb(
    xvobject object,
    kaddr client_data,
    kaddr call_data)
{
     double value;

     xvw_get_attribute(object, XVW_DOUBLE_VALUE, &value);
     kfprintf(kstderr, "Value = %g\n", value);
}
```

F. The Error Object

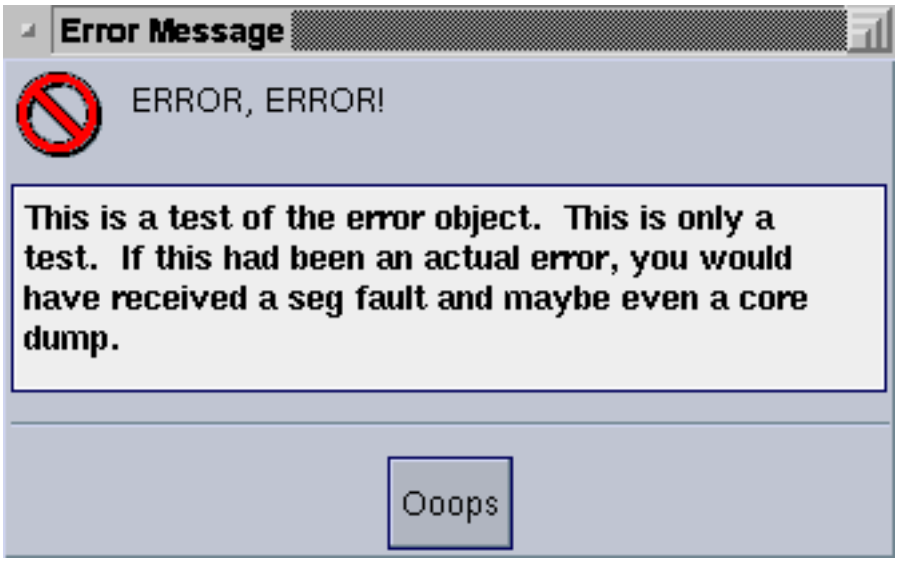


Figure 5: The error object is most often used indirectly, through *kerror()*, to print general information.

F.1. xvw_create_error() — *create an error object*

Synopsis

```
xvobject xvw_create_error(
    xvobject parent,
    char     *name)
```

Input Arguments

`parent`
: parent of the error widget; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The error object on success, NULL on failure

Description

The error object is a pop-up window that displays an error message. A stop sign icon on the upper left hand side of the error object draws the attention of the user; a single button on the upper right hand side of the error object allows the user to acknowledge the error message.

After the user clicks on the acknowledgment button, the error object is destroyed.

A callback can be installed on the error object, which will be fired when the user clicks on the acknowledgment button.

F.2. Attributes of the Error Object

Summary of Error Attributes

Attribute	Description
XVW_ERROR_BUTTON_LABEL	The label for the acknowledgment button.
XVW_ERROR_BUTTON_OBJECT	This *read-only* attribute allows you to obtain the the button object component of the error object (the button used to allow the user to acknowledge the error message).

XVW_ERROR_CALLBACK	If desired, *xvw_add_callback()* may be used to install a callback on the error object which will be fired when the user clicks on the acknowledgement button. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(errorobj, XVW_ERROR_CALLBACK,`
	`error_cb, client_data);`

XVW_ERROR_LABEL	The label for the error object.
XVW_ERROR_LABEL_OBJECT	This *read-only* attribute allows you to obtain the label object component of the error object (the object used to display the label).
XVW_ERROR_MESSAGE	The error message to be displayed in the error object.
XVW_ERROR_PIXMAP	This is the pixmap that appears to the upper left of the error object. Candidates for the value of this attribute may be created with the use of *XCreatePixmap()*; see *The Xlib Reference Manual* by O'Reilly and Associates. Note that this attribute is mutually exclusive with XVW_ERROR_PIXMAPFILE; specify one or the other, not both.
XVW_ERROR_PIXMAPFILE	This is the file defining the pixmap that appears at the upper left of the error object.
XVW_ERROR_TEXT_OBJECT	This *read-only* attribute allows you to obtain the the text object component of the error object (the text displaying the error message).

Descriptions of Error Attributes

Attribute (Resource Name)	Type	Default	Legal Values
XVW_ERROR_BUTTON_LABEL (N/A)	char *	"Ok"	any printable text

Descriptions of Error Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_ERROR_BUTTON_OBJECT (N/A)	xvobject	NULL	The button object (read-only)
XVW_ERROR_CALLBACK (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_ERROR_LABEL (N/A)	char *	NULL	any printable text
XVW_ERROR_LABEL_OBJECT (N/A)	xvobject	NULL	The label object (read-only).
XVW_ERROR_MESSAGE (N/A)	char *	NULL	any printable text
XVW_ERROR_PIXMAP (N/A)	Pixmap	The "stop sign" pixmap.	Valid Pixmap structure
XVW_ERROR_PIXMAPFILE (errorPixmapfile)	char *	The stopsign.xpm file in the *xvobjects/misc/pixmaps* directory, which defines the "stop sign" pixmap.	The full path to a valid xpm or xbm file, defining the desired pixmap. Note that the path may contain $TOOLBOX.
XVW_ERROR_TEXT_OBJECT (N/A)	xvobject	NULL	The text object (read-only)

F.3. Complete Resource Set of the Error Object

The complete resource set for the error object includes:

1. The error object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3,"The kwidgets Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object".

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

.section 2 "Example using the Error Object" An example program using the error object can be found in `$DESIGN/examples/xvobjects/error/example.c`.

```
#include <design.h>

/*
```

```
 *  This example creates an error object to display an error message.
 *
 *  IMPORTANT NOTE: in general, the error object should *not* be created
 *                  directly;  use of kerror() is the conventional way to
 *                  create messages in VisiQuest, so that there is standard
 *                  formatting enforced.  this example is really only for
 *                  academic purposes.
 */
main(
   int  argc,
   char *argv[])
{
     xvobject error;
     xvobject toplevel;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /*
      *  Create the error object. Note since the parent is NULL, a
      *  toplevel window will be created and the error object placed
      *  inside.   Set the label, the message, the button label.
      */
     error = xvw_create_error(NULL, "Error Message");
     xvw_set_attributes(error,
          XVW_ERROR_LABEL,   "ERROR, ERROR!",
          XVW_ERROR_MESSAGE, "This is a test of the error object.  This is only a test.  If th
          XVW_ERROR_BUTTON_LABEL, "Ooops",
          NULL);

     /* display and run the program.  */
     xvf_run_form();
}
```

G. The Float Object

Figure 6: The Float GUI object provides an float text and scrollbar window in which the user may enter an float value.

G.1. xvw_create_float() — *create a float object*

Synopsis

```
xvobject xvw_create_float(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: parent of the float object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The float GUI object on success, NULL on failure

Description

The float GUI object allows the user to enter a float precision value. It features a text box in which the user may explicitly enter the float value; the float value is registered when the user hits <cr>. The float value may be bounded by a minimum and maximum value, if desired.

A scroll bar provides an alternate way for the user to specify the float value. The float object consists of a manager object with four children: a label object, a text object, a scrollbar, and a pixmap object.

A callback can be installed on the float object, which will be fired when the user enters a new float value.

G.2. Attributes of the Float Object

Summary of Float Attributes	
Attribute	**Description**

XVW_FLOAT_CALLBACK	If desired, *xvw_add_callback()* may be used to install a callback may be installed on the float object which will be fired when the user enters a new float value, either by typing it in and pressing <cr>, or by using the scrollbar. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(floatobj, XVW_FLOAT_CALLBACK,`
	`float_cb, client_data);`
	Note that the current floating point value of the float object will be passed to the callback in the *call_data*. The value must be cast to a float pointer before use, as in:
	`float *value = (float *) call_data;`
	Alternatively, the float value may be obtained with XVW_FLOAT_VALUE.

XVW_FLOAT_CRLABEL_OBJECT	This *read-only* attribute allows you to obtain the pixmap object component of the the float object that indicates a "live" selection.
XVW_FLOAT_INCREMENT	The value to increment the scroll bar thumb.
XVW_FLOAT_LABEL	This is the text that will appear in the label object component of the float object. Provide text appropriate as a title of the float object.
XVW_FLOAT_LABEL_OBJECT	This *read-only* attribute allows you to obtain the label object component of the the float object.
XVW_FLOAT_MAXVALUE	The maximum float value allowed. If both XVW_FLOAT_MINVALUE and XVW_FLOAT_MAXVALUE are 0.0, the float value is unbounded.
XVW_FLOAT_MINVALUE	The minimum float value allowed. If both XVW_FLOAT_MINVALUE and XVW_FLOAT_MAXVALUE are 0.0, the float value is unbounded.
XVW_FLOAT_SCROLLBAR_OBJECT	This *read-only* attribute allows you to obtain the scrollbar object component of the the float object.
XVW_FLOAT_TEXT_OBJECT	This *read-only* attribute allows you to obtain the text object component of the the float object.
XVW_FLOAT_VALUE	The float precision value which is currently displayed in the text object. This attribute can be used to initialize the float value to be displayed in the text object, or to acquire a float value that has been entered by the user in the text object.

Descriptions of Float Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_FLOAT_CALLBACK (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_FLOAT_CRLABEL_OBJECT (N/A)	xvobject	NULL	The pixmap object (read-only).
XVW_FLOAT_INCREMENT (N/A)	float	calculated using size of scrollbar	values < (maxvalue - minvalue)/2
XVW_FLOAT_LABEL (N/A)	char *	NULL	any printable text
XVW_FLOAT_LABEL_OBJECT (N/A)	xvobject	NULL	The label object (read-only).
XVW_FLOAT_MAXVALUE (N/A)	float	1.0	any float value
XVW_FLOAT_MINVALUE (N/A)	float	0.0	any float value
XVW_FLOAT_SCROLLBAR_OBJECT (N/A)	xvobject	NULL	The scrollbar object (read-only).
XVW_FLOAT_TEXT_OBJECT (N/A)	xvobject	NULL	The text object (read-only).
XVW_FLOAT_VALUE (N/A)	float	0.0	any float value

G.3. Complete Resource Set of the Float Object

The complete resource set for the float object includes:

1. The float object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3,"The kwidgets Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object".

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

.section 2 "Example using the Float Object" An example program using the float object can be found in `$DESIGN/examples/xvobjects/float/example.c`.

```
#include <design.h>

/*
```

```
 *  This example creates a simple float GUI object, which may
 *  be used for allowing the user to enter a float.
 *
 *  A callback is installed on the float object so that when the
 *  user changes the value of the float either by entering a number
 *  in the text parameter box and hitting <cr> or by using the scrollbar,
 *  the current value is printed to the tty.
 *
 *  Note that the float object should *not* be created directly in an
 *  xvroutine, as use of the Float (-f) UIS line in the *.form file is
 *  both easier to use and a more standard use of the VisiQuest system.
 *  However, the float object is provided for use with hybrid xvroutines,
 *  (such as this example) which do not use a "formalized" GUI as defined
 *  in a *.form file.
 */
static void float_cb PROTO((xvobject, kaddr, kaddr));

main(
   int  argc,
   char *argv[])
{
     xvobject manager;
     xvobject object;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /* create a manager backplane to be a parent for the float object */
     manager = xvw_create_manager(NULL, "parent");
     xvw_set_attributes(manager,
                  XVW_WIDTH,       300,
                     XVW_HEIGHT,      100,
                  NULL);

     /*
         *  Create the float object.  give it a label, a default value,
         *  and bound it with a minimum value of 0 and a maximum value of
         *  255.  tack it horizontally to the parent so that it spans the
         *  width of the manager backplane.  center it in the middle of
         *  the parent.
      */
     object = xvw_create_float(manager, "float");
     xvw_set_attributes(object,
          XVW_FLOAT_LABEL,    "Float Number",
          XVW_FLOAT_VALUE,     123.456,
          XVW_FLOAT_MINVALUE,  0.0,
          XVW_FLOAT_MAXVALUE,  255.0,
          XVW_TACK_EDGE,       KMANAGER_TACK_HORIZ,
          XVW_ABOVE,     NULL,
          XVW_BELOW,     NULL,
          NULL);
     xvw_add_callback(object, XVW_FLOAT_CALLBACK, float_cb, NULL);
```

```
     /* display & run the program */
     xvf_run_form();
}

/*
 *  the callback for the float will be fired when the user changes the
 *  value of the float, either by using the scrollbar or by entering
 *  a value in the text parameter box & hitting <cr>.  this callback simply
 *  prints the current value of the float.
 */
static void float_cb(
    xvobject object,
    kaddr client_data,
    kaddr call_data)
{
     float value;

     xvw_get_attribute(object, XVW_FLOAT_VALUE, &value);
     kfprintf(kstderr, "Value = %g\n", value);
}
```

H. The Help Object

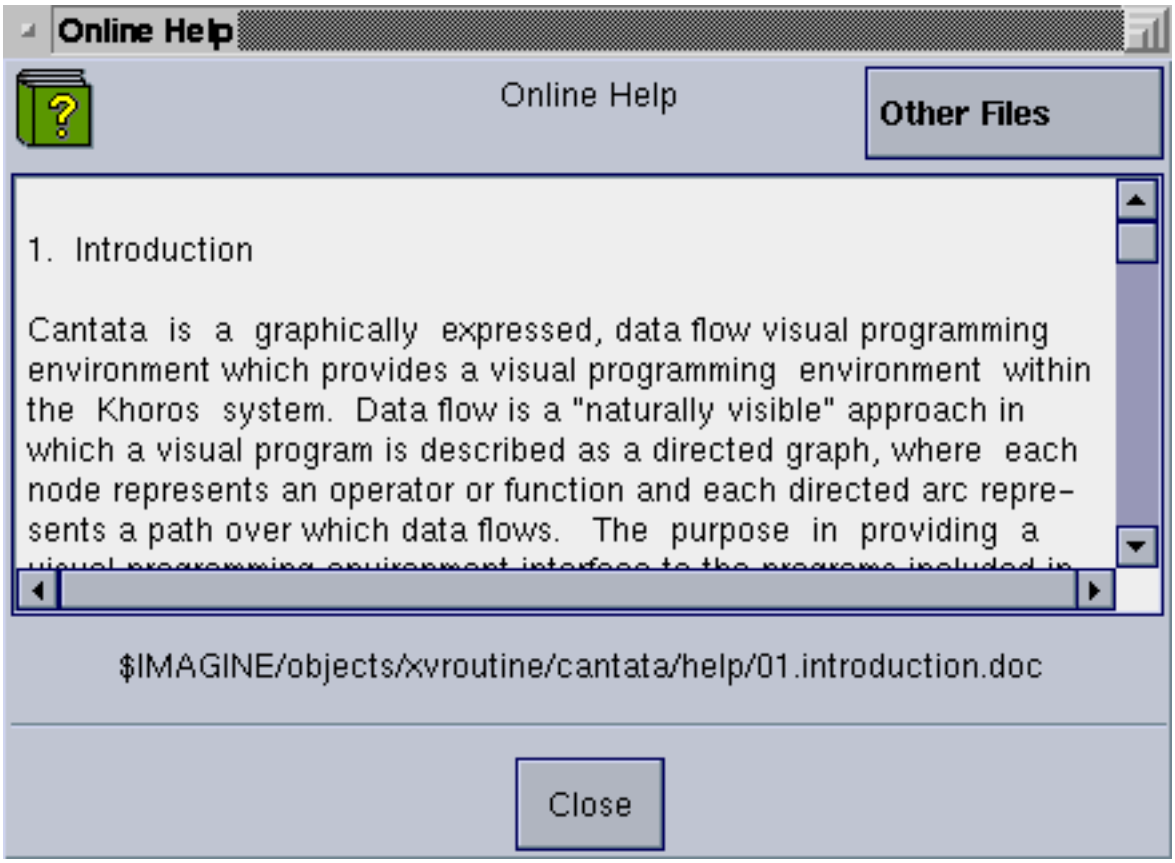


Figure 7: The help object is by all VisiQuest xvroutines to format and display help pages.

H.1. xvw_create_help() — *create a help object*

Synopsis

```
xvobject xvw_create_help(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: parent of the help object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The help widget on success, NULL on failure

Description

The help object provides a mechanism for displaying online help pages. Online help pages providing information corresponding to the man page of a software object are automatically generated as *.hlp files; xvroutines will have additional help pages created from scratch, named *.doc (see Chapter 6 of the Toolbox Programmer's Manual for more details).

The largest part of the help object is devoted to a large, scrollable window, in which the help page is displayed. On the top is a label for the help object; on the top right is a "Quit" button that is used to pop down the help object; on the top left is a pulldown menu that may be used to view other online help pages. On the bottom is a label listing the path to the help file being displayed.

The help object will format a help page that contain *roff* formatting commands before displaying it; alternatively, it can be used to display plain ascii text files as is. The help object may be provided with a path to a particular help file, or a path to a directory in which many help files exist. If the path provided is a path to a directory, the help page displayed by default will be the first one in the directory; if the path provided is a path to a particular file, that file will be displayed. In either case, if other help files (ie, files with names that end in ".doc" or ".hlp") will be accessable from the button on the upper left hand corner of the help object, labelled, "More Help Pages".

The first line in a help file controls the label of the entry in the "More Help Pages" pulldown menu that will cause that help file to be displayed. Such a line reads as follows:

```
.onlineHelp TOOLBOX "Label To Appear In Menu" ONAME
```

The word "TOOLBOX" should be replaced with the name of the toolbox, the word "ONAME" should be replaced with the program name, and the label that you would like the user to see when they using the "Other Files" pulldown menu should be entered instead of "Label To Appear In Menu". For example, the first line of the online help page for the "Files" subform of editimage reads:

```
.onlineHelp ENVISION "Input/Output" EDITIMAGE
```

H.2. Attributes of the Help Object

Summary of Help Attributes	
Attribute	**Description**

Summary of Help Attributes	
Attribute	**Description**
XVW_HELP_CALLBACK	If desired, *xvw_add_callback()* may be used to install a callback on the help object that will be fired when the user clicks on the "Quit" button. When calling *xvw_add_callback()*, pass this attribute directly, as in `xvw_add_callback(helpobj, XVW_HELP_CALLBACK, help_cb, client_data);` Note that this attribute does not apply if XVW_HELP_DISPLAYQUIT is set to FALSE.
XVW_HELP_DESTROY_ON_QUIT	If FALSE, the help object will not be destroyed when the user clicks on the "Quit" button, but will simply be unmapped.
XVW_HELP_DISPLAYICON	If FALSE, this attribute will suppress creation of the help icon.
XVW_HELP_DISPLAYMENU	If FALSE, this attribute will suppress creation of the options menu button.
XVW_HELP_DISPLAYQUIT	If FALSE, this attribute will suppress creation of the quit button.
XVW_HELP_DISPLAYTITLE	If FALSE, this attribute will suppress creation of the title.
XVW_HELP_FILENAME	The name of the help file to display. Specify the full path to the help file, using $*TOOLBOX* where *TOOLBOX* is the name of the toolbox containing the program.
XVW_HELP_ICON_OBJECT	This *read-only* attribute allows you to obtain the icon component of the help object.
XVW_HELP_INTERPRET_ROFF	If true, the help object will look for roff commands in the file, and call kman to format the file if any are present. If false, the help object will display the ascii text exactly as it is without formatting.
XVW_HELP_MENU_OBJECT	This *read-only* attribute allows you to obtain the menubutton component of the help object that provides access to other help files in the directory.
XVW_HELP_MORE_FILES	If FALSE, the help object will not stat the directory in order build the list of files included in the "Other Files" submenu.
XVW_HELP_NAME	This is the text that appears on the label object component that displays the filename.
XVW_HELP_NAME_OBJECT	This *read-only* attribute allows you to obtain the label component of the help object that displays the filename.
XVW_HELP_PIXMAP	This is the pixmap that appears to the upper left of the help object. Candidates for the value of this attribute may be created with the use of *XCreatePixmap()*; see *The Xlib Reference Manual* by O'Reilly and Associates. Note that this attribute is mutually exclusive with XVW_HELP_PIXMAPFILE; specify one or the other, not both.
XVW_HELP_PIXMAPFILE	This is the file defining the pixmap that appears at the upper left of the help object.
XVW_HELP_QUITLABEL	This is the text that appears on the quit button of the help object
XVW_HELP_QUIT_OBJECT	This *read-only* attribute allows you to obtain the button object component of the help object that lets the user quit.

Summary of Help Attributes	
Attribute	**Description**
XVW_HELP_TEXTDISPLAY_OBJECT	This *read-only* attribute allows you to obtain the textdisplay object component of the help object that does the actual display of online help text.
XVW_HELP_TITLE	This is the text that appears on the label object that displays the title of the help object.
XVW_HELP_TITLE_OBJECT	This *read-only* attribute allows you to obtain the label component of the help object that displays the title.

Descriptions of Help Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_HELP_CALLBACK (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_HELP_DESTROY_ON_QUIT (helpDestroyOnQuit)	int	TRUE	TRUE/FALSE
XVW_HELP_DISPLAYICON (helpDisplaymenu)	int	TRUE	TRUE/FALSE
XVW_HELP_DISPLAYMENU (helpDisplaymenu)	int	TRUE	TRUE/FALSE
XVW_HELP_DISPLAYQUIT (helpDisplayquit)	int	TRUE	TRUE/FALSE
XVW_HELP_DISPLAYTITLE (helpDisplaytitle)	int	TRUE	TRUE/FALSE
XVW_HELP_FILENAME (N/A)	char *	NULL	valid help file
XVW_HELP_ICON_OBJECT (N/A)	xvobject	NULL	The icon object (read-only)
XVW_HELP_INTERPRET_ROFF (helpInterpretRoff)	int	TRUE	TRUE/FALSE
XVW_HELP_MENU_OBJECT (N/A)	xvobject	NULL	The menubutton object (read-only)
XVW_HELP_MORE_FILES (helpMoreFiles)	int	TRUE	TRUE/FALSE
XVW_HELP_NAME (N/A)	char *	"No File Currently Being Displayed"	any printable text
XVW_HELP_NAME_OBJECT (N/A)	xvobject	NULL	The label object that displays the filename (read-only)

Descriptions of Help Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_HELP_PIXMAP (N/A)	Pixmap	The "help" pixmap.	Valid Pixmap structure
XVW_HELP_PIXMAPFILE (helpPixmapfile)	char *	The help.xpm file in the *xvobjects/misc/pixmaps* directory, which defines the "help" pixmap.	The full path to a valid xpm or xbm file, defining the desired pixmap. Note that the path may contain $TOOLBOX.
XVW_HELP_QUITLABEL (N/A)	char *	"Quit"	Any printable text
XVW_HELP_QUIT_OBJECT (N/A)	xvobject	NULL	The button object that lets the user quit (read-only)
XVW_HELP_TEXTDISPLAY_OBJECT (N/A)	xvobject	NULL	The textdisplay object (read-only).
XVW_HELP_TITLE (N/A)	char *	"Online Help"	any printable text
XVW_HELP_TITLE_OBJECT (N/A)	xvobject	NULL	The label object that displays the title (read-only)

H.3. Complete Resource Set of the Help Object

The complete resource set for the help object includes:

1. The help object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3,"The kwidgets Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object".

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

H.4. Example using the Help Object

An example program using the help object can be found in `$DESIGN/examples/xvobjects/help/example.c`.

```
#include <design.h>

/*
 *  The program demonstrates the help object.  The help object displays
 *  a help file which is formatted with "roff" commands.  The commands
 *  are interpreted, and the formatted output displayed.
```

```
 *
 *  % example
 *             displays the default help files in the "guise" help directory
 *
 *  % example {directory}
 *             displays the help files in the directory specified
 *
 *  IMPORTANT NOTE: this example will not work properly if you do not have
 *                  groff, gtbl, and geqn installed on your system.
 *
 *  ALSO NOTE: the help object takes a while to display. be patient.
 */
main(
   int  argc,
   char *argv[])
{
        xvobject help;
        char     *filename = "./Capture.doc";

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

        if (argc > 1)
           filename = argv[1];

        /* initialize the xvwidgets lib */
        if (!xvw_initialize(XVW_MENUS_XVFORMS))
        {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
        }

        /*
         *  Create the help object. Note that since the parent is NULL, a
         *  toplevel window will be created and the text displayed
         *  inside.  The XVW_HELP_FILENAME attribute is used to specify
         *  the directory in which the help file(s) exist.  Note that help
         *  file(s) should be formatted with "roff" commands, which will be
         *  interpreted.
         */
        help = xvw_create_help(NULL, "Online Help");
        xvw_set_attributes(help,
                XVW_HELP_TITLE,     "Boy!  You sure do need help",
                XVW_HELP_FILENAME,   filename,
                XVW_HELP_QUITLABEL, "Quit!",
                NULL);
     xvw_set_attribute(help, XVW_MAXIMUM_HEIGHT, 350);

     /* display & run the program.  */
        xvf_run_form();
}
```

I. The Info Object

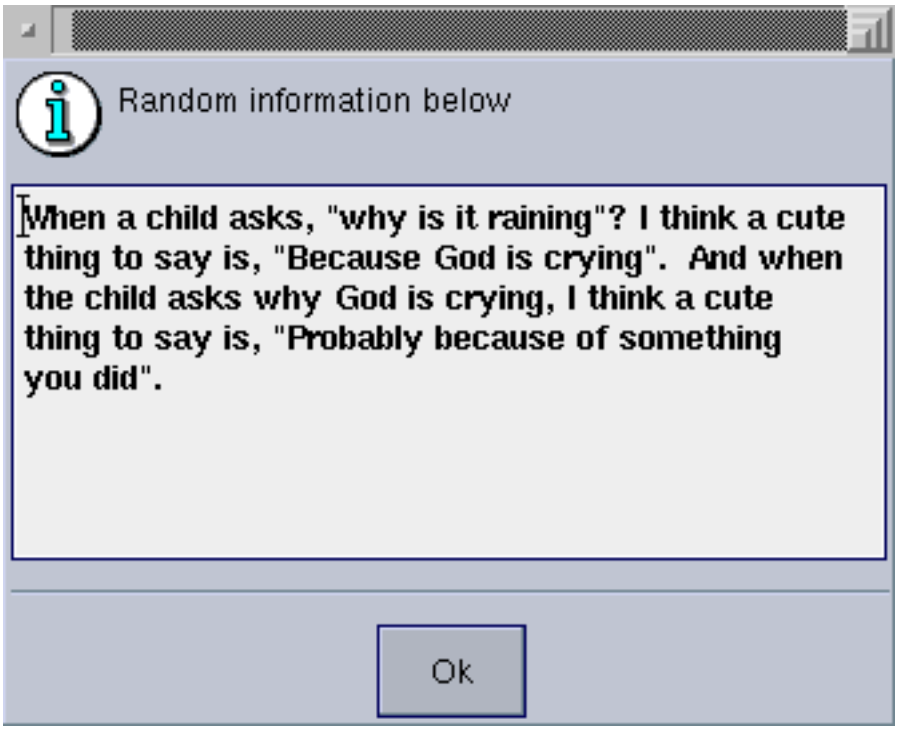


Figure 8: The info object is most often used indirectly, through *kinfo()*, to print general information.

I.1. xvw_create_info() — *create an info object*

Synopsis

```
xvobject xvw_create_info(
    xvobject parent,
    char     *name)
```

Input Arguments

`parent`
: parent of the info object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The info object on success, NULL on failure

Description

The info object is a pop-up window that displays an informative message. A "nose knows" icon on the upper left hand side of the info object draws the attention of the user; a single button on the upper right hand side of the info object allows the user to acknowledge the information.

After the user clicks on the acknowledgment button, the info object is destroyed.

A callback can be installed on the info object, which will be fired when the user clicks on the acknowledgment button.

I.2. Attributes of the Info Object

Summary of Icon Attributes

Attribute	Description
XVW_INFO_BUTTON_LABEL	The label of the acknowledgement button on the info object
XVW_INFO_BUTTON_OBJECT	This *read-only* attribute allows you to obtain the the button object component of the info object (the button used to allow the user to acknowledge the message).

XVW_INFO_CALLBACK	If desired, *xvw_add_callback()* may be used to install a callback on the info object which will be fired when the user clicks on the acknowledgement button. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(info_obj, XVW_INFO_CALLBACK,`
	`info_cb, client_data);`

XVW_INFO_LABEL	The label to be displayed on the info object.
XVW_INFO_LABEL_OBJECT	This *read-only* attribute allows you to obtain the label object component of the the info object (the object used to display the label).
XVW_INFO_MESSAGE	The message to be displayed in the info object.
XVW_INFO_PIXMAP	This is the pixmap that appears to the upper left of the info object. Candidates for the value of this attribute may be created with the use of *XCreatePixmap()*; see *The Xlib Reference Manual* by O'Reilly and Associates. Note that this attribute is mutually exclusive with XVW_INFO_PIXMAPFILE; specify one or the other, not both.
XVW_INFO_PIXMAPFILE	This is the file defining the pixmap that appears at the upper left of the info object.
XVW_INFO_TEXT_OBJECT	This *read-only* attribute allows you to obtain the the text object component of the info object (the text displaying the message).

Descriptions of Info Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_INFO_BUTTON_LABEL (N/A)	char *	"Ok"	any printable text
XVW_INFO_BUTTON_OBJECT (N/A)	xvobject	NULL	The button object (read-only)
XVW_INFO_CALLBACK (N/A)	void (*call-back_rou-tine)(xvob-ject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_INFO_LABEL (N/A)	char *	"Info:"	any printable text
XVW_INFO_LABEL_OBJECT (N/A)	xvobject	NULL	The label object (read-only).
XVW_INFO_MESSAGE (N/A)	char *	NULL	any printable text
XVW_INFO_PIXMAP (N/A)	Pixmap	The "nose knows" pixmap.	Valid Pixmap structure
XVW_INFO_PIXMAPFILE (infoPixmapfile)	char *	The info.xpm file in the *xvob-jects/misc/pixmaps* directory, which defines the "nose knows" pixmap.	The full path to a valid xpm or xbm file, defining the desired pixmap. Note that the path may contain $TOOLBOX.
XVW_INFO_TEXT_OBJECT (N/A)	xvobject	NULL	The text object (read-only)

I.3. Complete Resource Set of the Info Object

The complete resource set for the info object includes:

1. The info object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3,"The kwidgets Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object".

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

I.4. Example using the Info Object

An example program using the info object can be found in `$DESIGN/examples/xvobjects/info/example.c`.

```
#include <design.h>

/*
 *  This example creates an info object to display information.
 *
 *  IMPORTANT NOTE: in general, the info object should *not* be created
 *                  directly;  use of kinfo() is the conventional way to
 *                  create messages in VisiQuest, so that there is standard
 *                  formatting enforced.  this example is really only for
 *                  academic purposes.
 */
void main(
   int  argc,
   char *argv[])
{
     xvobject info;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /*
      *  Create the info object. Note that since the parent is NULL, a
      *  toplevel window will be created and the info object placed
      *  inside.  Set the label, the information message, and the button
         *  label.
      */
     info = xvw_create_info(NULL, " ");
     xvw_set_attributes(info,
          XVW_INFO_MESSAGE,   "When a child asks,
          XVW_INFO_LABEL, "Random information below",
          NULL);

     /* display & run the program */
     xvf_run_form();
}
```

J. The Inputfile Object

Figure 9: The InputFile GUI object provides an input file window in which the user may enter an input filename.

J.1. xvw_create_inputfile() — *create a inputfile GUI object*

Synopsis

```
xvobject xvw_create_inputfile(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: parent of the inputfile object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The inputfile object on success, NULL on failure

Description

The inputfile GUI object allows the user to enter an input file. It features a text box in which the user may type in the filename; the filename is registered when the user hits <cr>. The label of the inputfile GUI object is really a button, which may be used to display the file browser. The file browser provides an alternate way for input file selection.

The inputfile GUI object consists of a manager object with three children: a button object, a text object, and a pixmap object.

A callback can be installed on the inputfile object, which will be fired when the user enters a new filename.

J.2. Attributes of the Inputfile Object

Summary of InputFile Attributes	
Attribute	**Description**
XVW_INPUTFILE_BROWSER_OBJECT	This *read-only* attribute allows you to obtain the browser object which is popped up when the user clicks on the button of the inputfile object.
XVW_INPUTFILE_BUTTON_OBJECT	This *read-only* attribute allows you to obtain the button object component of the the inputfile object.

XVW_INPUTFILE_CALLBACK	If desired, *xvw_add_callback()* may be used to install a callback on the inputfile object which will be fired when the user enters a new filename, either by typing it in and pressing <cr>, or by using the browser. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(inputfileobj, XVW_INPUTFILE_CALLBACK, inputfile_cb, client_data);`
	Note that the current filename of the input file object will be passed to the callback in the *call_data*. The value must be cast to a string before use, as in:
	`char *filename = *((char **) call_data);`
	Alternatively, the filename may be obtained with XVW_INPUTFILE_FILENAME.

XVW_INPUTFILE_CRLABEL_OBJECT	This *read-only* attribute allows you to obtain the pixmap object component of the the inputfile object that indicates a "live" selection.
XVW_INPUTFILE_DISPLAY_BUTTON	This attribute controls whether the browser button should be displayed (mapped) or not. If not then the text object
XVW_INPUTFILE_FILENAME	The filename which is currently displayed in the text object. This attribute can be used to initialize the filename to be displayed in the inputfile object, or to acquire the filename that has been entered by the user in the text object.
XVW_INPUTFILE_LABEL	This is the text that will appear in the label object component of the inputfile object. Provide text appropriate as a title of the inputfile object.
XVW_INPUTFILE_TEXT_OBJECT	This *read-only* attribute allows you to obtain the text object component of the the inputfile object.

Descriptions of InputFile Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_INPUTFILE_BROWSER_OBJECT (N/A)	xvobject	NULL	The browser object (read-only).
XVW_INPUTFILE_BUTTON_OBJECT (N/A)	xvobject	NULL	The button object (read-only).

Descriptions of InputFile Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_INPUTFILE_CALLBACK (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_INPUTFILE_CRLABEL_OBJECT (N/A)	xvobject	NULL	The pixmap object (read-only).
XVW_INPUTFILE_DISPLAY_BUTTON (inputfileDisplayButton)	int	TRUE	TRUE/FALSE
XVW_INPUTFILE_FILENAME (N/A)	char *	NULL	any valid input file name
XVW_INPUTFILE_LABEL (N/A)	char *	NULL	any printable text
XVW_INPUTFILE_TEXT_OBJECT (N/A)	xvobject	NULL	The text object (read-only).

J.3. Complete Resource Set of the InputFile Object

The complete resource set for the inputfile object includes:

1. The inputfile object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3,"The kwidgets Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object".

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

.section 2 "Example using the InputFile Object" An example program using the inputfile object can be found in `$DESIGN/examples/xvobjects/inputfile/example.c`.

```
#include <design.h>

/*
 *  This example creates a simple inputfile GUI object, which may
 *  be used for allowing the user to enter a input file.  The user may
 *  enter a value in the text parameter box, or click on the label button,
 *  which will bring up the file browser from which a file may be picked.
 *
 *  A callback is installed on the input file object so that when the
 *  user changes the value of the input file, the current
 *  filename is printed to the tty.
 *
```

```
 *  Note that the input file object should *not* be created directly in an
 *  xvroutine, as use of the InputFile (-I) UIS line in the *.form file is
 *  both easier to use and a more standard use of the VisiQuest system.
 *  However, the inputfile object is provided for use with hybrid xvroutines,
 *  (such as this example) which do not use a "formalized" GUI as defined
 *  in a *.form file.
 */
static void inputfile_cb PROTO((xvobject, kaddr, kaddr));

main(
   int  argc,
   char *argv[])
{
     xvobject manager;
     xvobject object;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /* create a manager backplane for the inputfile object */
     manager = xvw_create_manager(NULL, "parent");
     xvw_set_attributes(manager,
                  XVW_WIDTH,       300,
                     XVW_HEIGHT,      100,
                  NULL);

     /*
         *  Create the inputfile object.  give it a label.
         *  tack it horizontally to the parent so that it spans the
         *  width of the manager backplane.  center it in the middle of
         *  the parent.
      */
     object = xvw_create_inputfile(manager, "inputfile");
     xvw_set_attributes(object,
          XVW_INPUTFILE_LABEL,     "Input File",
          XVW_INPUTFILE_FILENAME,  "image:ball",
          XVW_TACK_EDGE,           KMANAGER_TACK_HORIZ,
          XVW_ABOVE,          NULL,
          XVW_BELOW,          NULL,
          NULL);
     xvw_add_callback(object, XVW_INPUTFILE_CALLBACK, inputfile_cb, NULL);

     /* display & run the program */
     xvf_run_form();
}

/*
 *  the callback for the inputfile will be fired when the user changes the
 *  value of the input file & hits <cr>, or when they use the browser to choose
 *  a file.  this callback simply prints the current filename.
 */
static void inputfile_cb(
```

```
    xvobject object,
    kaddr client_data,
    kaddr call_data)
{
     char *filename;

     xvw_get_attribute(object, XVW_INPUTFILE_FILENAME, &filename);
     kfprintf(kstderr, "Filename = %s\n", filename);
}
```

K. The Integer Object

Figure 10: The Integer GUI object provides an integer text and scrollbar window in which the user may enter an integer value.

K.1. xvw_create_integer() — *create an integer GUI object*

Synopsis

```
xvobject xvw_create_integer(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: parent of the integer object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The integer object on success, NULL on failure

Description

The integer GUI object allows the user to enter a integer value. It features a text box in which the user may explicitly enter the integer value; the integer value is registered when the user hits <cr>. The integer value may be bounded by a minimum and maximum value, if desired.

A scroll bar provides an alternate way for the user to specify the integer value. The integer object consists of a manager object with four children: a label object, a text object, a scrollbar, and a pixmap object.

A callback can be installed on the integer object, which will be fired when the user enters a new integer value.

K.2. Attributes of the Integer Object

Summary of Integer Attributes	
Attribute	**Description**

XVW_INTEGER_CALLBACK	If desired, *xvw_add_callback()* may be used to install a callback may be installed on the integer object which will be fired when the user enters a new int value, either by typing it in and pressing <cr>, or by using the scrollbar. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(intobj, XVW_INTEGER_CALLBACK, int_cb, client_data);`
	Note that the current value of the integer object will be passed to the callback in the *call_data*. The value must be cast to an integer pointer before use, as in:
	`int *value = (int *) call_data;`
	Alternatively, the int value may be obtained with XVW_INTEGER_VALUE.

XVW_INTEGER_CRLABEL_OBJECT	This *read-only* attribute allows you to obtain the pixmap object component of the the integer object that indicates a "live" selection.
XVW_INTEGER_INCREMENT	The value to increment the scroll bar thumb.
XVW_INTEGER_LABEL	This is the text that will appear in the label object component of the integer object. Provide text appropriate as a title of the integer object.
XVW_INTEGER_LABEL_OBJECT	This *read-only* attribute allows you to obtain the label object component of the the integer object.
XVW_INTEGER_MAXVALUE	The maximum integer value allowed. If both XVW_INTEGER_MINVALUE and XVW_INTEGER_MAXVALUE are 0, the integer value is unbounded.
XVW_INTEGER_MINVALUE	The minimum integer value allowed. If both XVW_INTEGER_MINVALUE and XVW_INTEGER_MAXVALUE are 0.0, the integer value is unbounded.
XVW_INTEGER_SCROLLBAR_OBJECT	This *read-only* attribute allows you to obtain the scrollbar object component of the the integer object.
XVW_INTEGER_TEXT_OBJECT	This *read-only* attribute allows you to obtain the text object component of the the integer object.
XVW_INTEGER_VALUE	The integer precision value which is currently displayed in the text object. This attribute can be used to initialize the integer value to be displayed in the text object, or to acquire a integer value that has been entered by the user in the text object.

Descriptions of Integer Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**

Descriptions of Integer Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_INTEGER_CALLBACK (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_INTEGER_CRLABEL_OBJECT (N/A)	xvobject	NULL	The pixmap object (read-only).
XVW_INTEGER_INCREMENT (N/A)	int	calculated using size of scrollbar	values < (maxvalue - minvalue)/2
XVW_INTEGER_LABEL (N/A)	char *	NULL	any printable text
XVW_INTEGER_LABEL_OBJECT (N/A)	xvobject	NULL	The label object (read-only).
XVW_INTEGER_MAXVALUE (N/A)	int	1.0	any integer value
XVW_INTEGER_MINVALUE (N/A)	int	0.0	any integer value
XVW_INTEGER_SCROLLBAR_OBJECT (N/A)	xvobject	NULL	The scrollbar object (read-only).
XVW_INTEGER_TEXT_OBJECT (N/A)	xvobject	NULL	The text object (read-only).
XVW_INTEGER_VALUE (N/A)	int	0.0	any integer value

K.3. Complete Resource Set of the Integer Object

The complete resource set for the integer object includes:

1. The integer object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3,"The kwidgets Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object."

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

K.4. Example using the Integer Object

An example program using the integer object can be found in `$DESIGN/examples/xvobjects/integer/example.c`.

```
#include <design.h>

/*
 *  This example creates a simple integer GUI object, which may
 *  be used for allowing the user to enter a integer.
 *
 *  A callback is installed on the integer object so that when the
 *  user changes the value of the integer either by entering a number
 *  in the text parameter box and hitting <cr> or by using the scrollbar,
 *  the current value is printed to the tty.
 *
 *  Note that the integer object should *not* be created directly in an
 *  xvroutine, as use of the Integer (-i) UIS line in the *.form file is
 *  both easier to use and a more standard use of the VisiQuest system.
 *  However, the integer object is provided for use with hybrid xvroutines,
 *  (such as this example) which do not use a "formalized" GUI as defined
 *  in a *.form file.
 */
static void integer_cb PROTO((xvobject, kaddr, kaddr));

main(
   int  argc,
   char *argv[])
{
     xvobject manager;
     xvobject object;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /* create a manager backplane to be a parent for the integer object */
     manager = xvw_create_manager(NULL, "parent");
     xvw_set_attributes(manager,
                  XVW_WIDTH,       300,
                     XVW_HEIGHT,      100,
                  NULL);

     /*
         *  Create the integer object.  give it a label, a default value,
         *  and bound it with a minimum value of 0 and a maximum value of
         *  255.  tack it horizontally to the parent so that it spans the
         *  width of the manager backplane.  center it in the middle of
         *  the parent.
      */
     object = xvw_create_integer(manager, "integer");
     xvw_set_attributes(object,
          XVW_INTEGER_LABEL,    "Integer",
```

```
            XVW_INTEGER_VALUE,     12,
            XVW_INTEGER_MINVALUE,  0,
            XVW_INTEGER_MAXVALUE,  100,
            XVW_TACK_EDGE,         KMANAGER_TACK_HORIZ,
            XVW_ABOVE,       NULL,
            XVW_BELOW,       NULL,
            NULL);
     xvw_add_callback(object, XVW_INTEGER_CALLBACK, integer_cb, NULL);

     /* display & run the program */
     xvf_run_form();
}

/*
 *  the callback for the integer will be fired when the user changes the
 *  value of the integer, either by using the scrollbar or by entering
 *  a value in the text parameter box & hitting <cr>.  this callback simply
 *  prints the current value of the int.
 */
static void integer_cb(
    xvobject object,
    kaddr client_data,
    kaddr call_data)
{
     int value;

     xvw_get_attribute(object, XVW_INTEGER_VALUE, &value);
     kfprintf(kstderr, "Value = %d\n", value);
}
```

L. The Layout Object

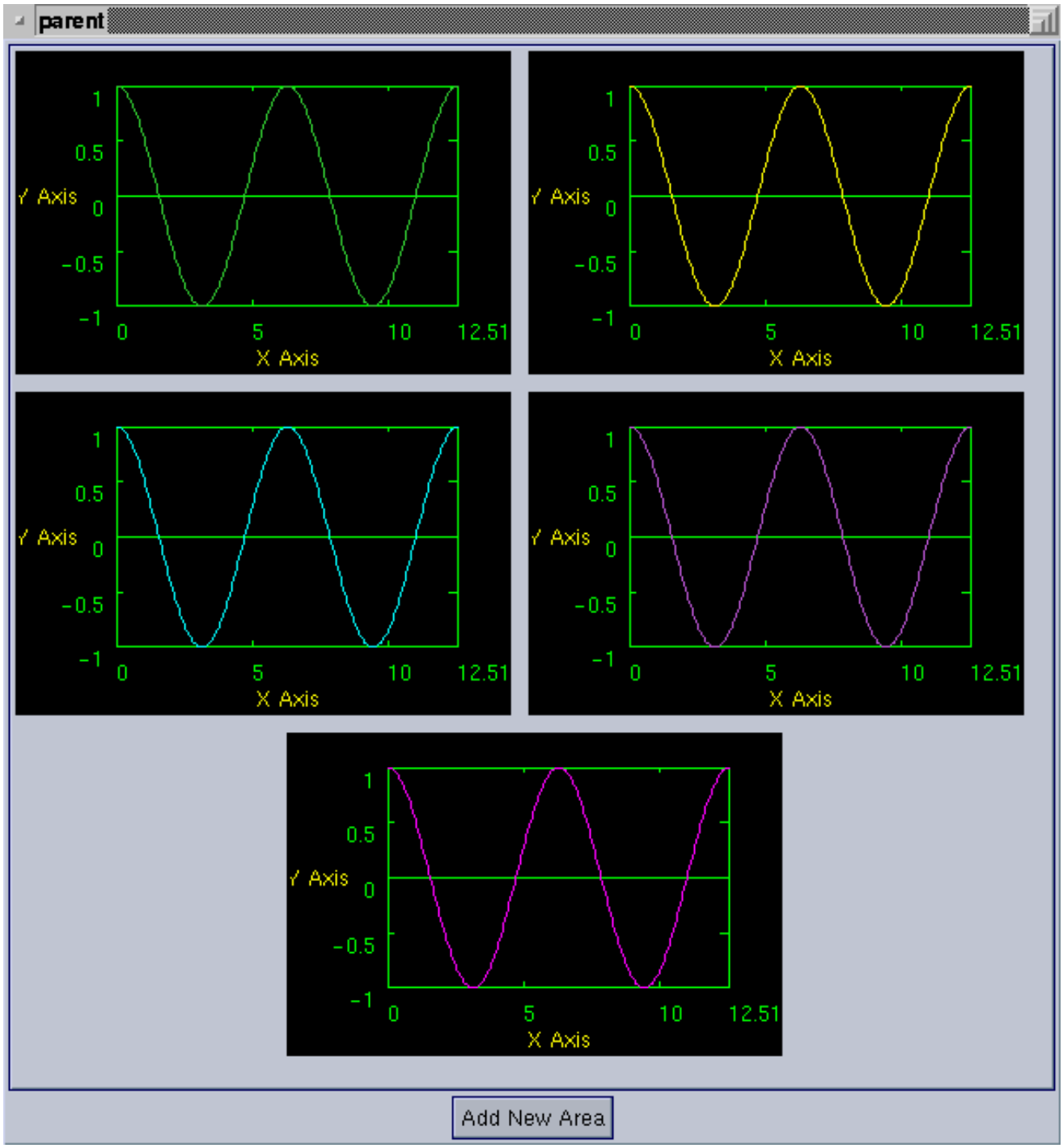


Figure 11: Here, the layout object is used to lay out multiple area objects, where each area object contains a plot.

L.1. xvw_create_layout() — *create a layout object*

Synopsis

```
xvobject xvw_create_layout(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent of the layout object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The layout object on success, NULL on failure

Description

The layout object is designed for doing layout of objects that are subclassed from the Manager widget. Only objects subclassed from the manager widget (*not* the manager object) may be created as children of the layout object. Such objects include area objects, image objects, zoom objects, viewport objects, and so on. See Chapter 1, "Introduction", of the Programming Services Manual, Volume 3, for diagrams depicting the objects that are subclassed from the Manager widget.

The layout object allows you to do quick and easy layout when a variety of such objects share a common backplane. You may specify the number of objects that should appear in a single row; relative layout specifications are not needed.

The layout object is especially effective when laying out objects *of the same type*, as it will preserve proportionality between the objects. It is often used in applications such as xprism, where the user will be interactively creating new objects to display data. The application need not implement special code to do appropriate layout of new objects as they are created, since the layout object does it automatically according to initial specifications.

L.2. Attributes of the Layout Object

Summary of Layout Attributes	
Attribute	**Description**

Summary of Layout Attributes	
Attribute	**Description**
XVW_LAYOUT_AREA_JUSTIFICATION	This attribute indicates how to lay out the last row of area objects, assuming that the last row has a smaller number of area objects than any of the other rows. For example, assume that there are 3 area objects, and XVW_NUMBER_ACROSS has been specified as 2. In this scenario, the layout object will place the first two area objects in the first row, and have a last area object to place somewhere in the second row. The XVW_AREA_JUSTIFICATION attribute indicates where in the last row the last area object will be placed. The following settings may be used: KLAYOUT_AREA_CENTER- center the area object(s) in the last row KLAYOUT_AREA_RIGHT- right justify the area object(s) in the last row KLAYOUT_AREA_LEFT- left justify the area object(s) in the last row KLAYOUT_AREA_FULL- "stretch" the area object(s) to fill the last row
XVW_LAYOUT_BORDER_SIZE	When the user selects one of the area object children of the the layout object, the border of the area object will be highlighted. This attribute specifies the line width of the highlighted border.
XVW_LAYOUT_BUFFER_SIZE	This attribute specifies the distance between area objects, in pixels.

XVW_LAYOUT_CALLBACK	If desired, *xvw_add_callback()* may be used to install a callback on the layout object which will be fired when the user selects the layout object by clicking on it. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(layoutobj, XVW_LAYOUT_CALLBACK,`
	`layout_cb, client_data);`

XVW_LAYOUT_NUMBER_ACROSS	This attribute specifies how many area objects will be placed on the same row before a new row is started.
XVW_LAYOUT_SELECTED_CHILD	Specifies which area object is currently selected; the selected area object will be highlighted. The default is NULL.

Descriptions of Layout Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_LAYOUT_AREA_JUSTIFICATION (layoutAreaJustification)	int	KLAYOUT_AREA_CENTER	KLAYOUT_AREA_CENTER KLAYOUT_AREA_RIGHT KLAYOUT_AREA_LEFT KLAYOUT_AREA_FULL
XVW_LAYOUT_BORDER_SIZE (layoutBorderSize)	int	2	values >= 0

Descriptions of Layout Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_LAYOUT_BUFFER_SIZE (layoutBufferSize)	int	5	values >= 0
XVW_LAYOUT_CALLBACK (N/A)	void (*call-back_rou-tine)(xvob-ject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_LAYOUT_NUMBER_ACROSS (layoutNumberAcross)	int	2	values > 0
XVW_LAYOUT_SELECTED_CHILD (N/A)	xvobject	NULL	valid xvobject

L.3. Complete Resource Set of the Layout Object

The Complete Resource Set for the layout object includes:

1. The layout object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3,"The kwidgets Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object".

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

L.4. Example using the Layout Object

An example program using the canvas object can be found in `$ENVISION/examples/plot/10.layout/example.c`.

M. The NotifyWindow Object

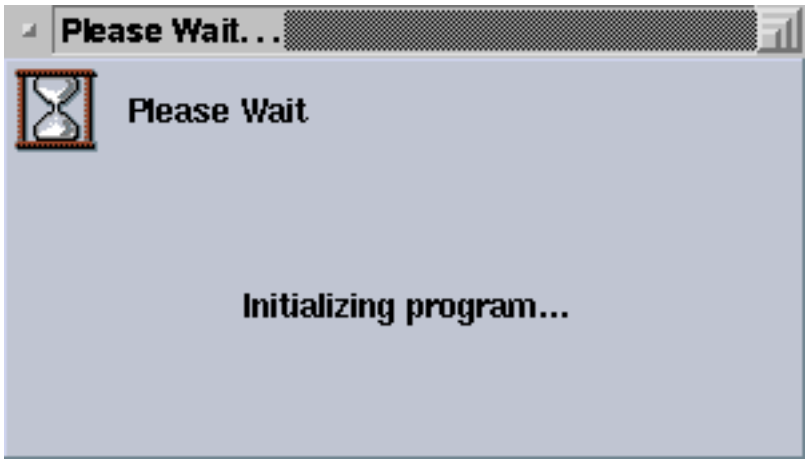


Figure 12: The NotifyWindow GUI object provides a notification window in which to convery certain messages to the user. The notify window can be used as a working area in which the messages can be updated.

M.1. xvw_create_notifywindow() — *create a notifywindow object*

Synopsis

```
xvobject xvw_create_notifywindow(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: parent of the notifywindow object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The notifywindow object on success, NULL on failure

Description

The notifywindow object provides a convenient mechanism for the application to notify the user when it is performing an operation that will take some time, and that the user is expected to wait patiently until it is done.

The notifywindow object pops up without placement; it contains an icon (by default, an hourglass), a

label, and a message so that the application may inform the user exactly why they are waiting.

With applications that will use the notifywindow periodically, it is recommended that a single notifywindow be created for the application on startup, with XVW_NOTIFYWINDOW_VISIBLE set to FALSE. Then, whenever necessary, the message displayed by the notifywindow can be updated using the XVW_NOTIFYWINDOW_MESSAGE attribute, and the notifywindow object popped up using the XVW_NOTIFYWINDOW_VISIBLE attribute. When the application is done, XVW_NOTIFYWINDOW_VISIBLE can be set to FALSE again, until the next time the notifywindow object is to be popped up.

M.2. Attributes of the NotifyWindow Object

Summary of NotifyWindow Attributes

Attribute	Description
XVW_NOTIFYWINDOW_ICON_OBJECT	This *read-only* attribute allows you to obtain the the icon object component of the notifywindow object (the icon displays the picture of the hourglass (or whatever you specify) on the notifywindow object). Note that by getting the icon object with this attribute, you may then use the XVW_ICON_PIXMAPFILE attribute with the icon object to specify another icon; alternatively, the resource name may be used to specify the desired pixmap in the app-defaults file, as in: *notifywindowIconObject.iconPixmapfile: my_pixmap.xpm
XVW_NOTIFYWINDOW_LABEL	The label that appears at the top of the notify window object. This is usually a generalized label such as "Working..." or some other string to indicate that the program is busy.
XVW_NOTIFYWINDOW_LABEL_OBJECT	This *read-only* attribute allows you to obtain the label object component of the the notifywindow object (the object used to display the label).
XVW_NOTIFYWINDOW_MESSAGE	This is the message that appears in the notify window while it is displayed; users are expected to read it while they are waiting for the notify window to go away. The message is typically used to indicate what the program is doing that is forcing the user to wait. With programs such as craftsman, for example, the message might inform the user that a new toolbox or a new software object is being created.
XVW_NOTIFYWINDOW_NOTIFYFOR	The XVW_NOTIFYWINDOW_NOTIFYFOR attribute is used to specify the object for which the notify window appears. Typically, this is the application's toplevel object. When a notify window is made "visible", the notify window will be centered around the specified object. If this field is not specified, then the notify window will be placed according to the location of the user's pointer.

Summary of NotifyWindow Attributes	
Attribute	**Description**
XVW_NOTIFYWINDOW_PIXMAP	This is the pixmap that appears in the notify window object. Candidates for the value of this attribute may be created with the use of *XCreatePixmap()*; see *The Xlib Reference Manual* by O'Reilly and Associates. Note that this attribute is mutually exclusive with XVW_NOTIFYWINDOW_PIXMAPFILE; specify one or the other, not both.
XVW_NOTIFYWINDOW_PIXMAPFILE	This is the file defining the pixmap that appears in the notify window.
XVW_NOTIFYWINDOW_TEXT_OBJECT	This *read-only* attribute allows you to obtain the text object component of the the notifywindow object (the object used to display the message that the user reads while they are waiting for the notifywindow to go away).
XVW_NOTIFYWINDOW_TITLE	The title that appears in the titlebar of the notify window object. Typically, this is the name of the application using the notify window. This title is only used if the user's window manager applies a window dressing to transient windows.
XVW_NOTIFYWINDOW_VISIBLE	Defines whether or not the notify window should be actively displayed. If the XVW_NOTIFYWINDOW_VISIBLE attribute is FALSE, then the notify window will be unmapped from the screen. If TRUE then the notify window will be placed centered according to the XVW_NOTIFYWINDOW_VISIBLE attribute.

Descriptions of NotifyWindow Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_NOTIFYWINDOW_ICON_OBJECT (N/A)	xvobject	NULL	The icon object (read-only).
XVW_NOTIFYWINDOW_LABEL (notifywindowLabel)	char *	NULL	any printable text
XVW_NOTIFYWINDOW_LABEL_OBJECT (N/A)	xvobject	NULL	The label object (read-only).
XVW_NOTIFYWINDOW_MESSAGE (N/A)	char *	NULL	any printable text.
XVW_NOTIFYWINDOW_NOTIFYFOR (N/A)	xvobject	NULL	Toplevel object of calling application.
XVW_NOTIFYWINDOW_PIXMAP (N/A)	Pixmap	The "hourglass" pixmap	Valid Pixmap structure
XVW_NOTIFYWINDOW_PIXMAPFILE (notifywindowPixmapfile)	char *	The hourglass.xbm file in the *xvobjects/misc/pixmaps* directory, which defines the "hourglass" pixmap.	The full path to a valid xpm or xbm file, defining the desired pixmap. Note that the path may contain $TOOLBOX.

Descriptions of NotifyWindow Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_NOTIFYWINDOW_TEXT_OBJECT (N/A)	xvobject	NULL	The text object (read-only).
XVW_NOTIFYWINDOW_TITLE (notifywindowTitle)	char *	NULL	any printable text.
XVW_NOTIFYWINDOW_VISIBLE (N/A)	int	FALSE	TRUE/FALSE

M.3. Complete Resource Set of the NotifyWindow Object

The complete resource set for the notifywindow object includes:

1. The notifywindow object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3,"The kwidgets Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object."

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

.section 2 "Example using the NotifyWindow Object"

An example program using the notifywindow object can be found in `$DESIGN/examples/xvobjects/notify/example.c`.

```
#include <design.h>

/*
 *  This example demonstrates the use of the notifywindow object.
 *  Notifywindows are nice when an application has to do something that
 *  will take a while, and the developer wants to inform the user about
 *  what's happening.  They don't require acknowledgement;  they just
 *  pop up to say something while the program is working, and go away
 *  when the program is done doing whatever it's doing.
 */

static void button_cb PROTO((xvobject, kaddr, kaddr));
static void quit_cb   PROTO((xvobject, kaddr, kaddr));

main(
   int  argc,
   char *argv[])
{
     xvobject manager;
        xvobject button;
        xvobject quit;
        xvobject notifywindow;
```

```
   /* initialize VisiQuest program */
   khoros_initialize(argc, argv, "DESIGN");

/* initialize the xvwidgets lib */
if (!xvw_initialize(XVW_MENUS_XVFORMS))
{
   kerror(NULL, "main", "unable to open display");
   kexit(KEXIT_FAILURE);
}

/* create a manager object backplane for a button */
manager = xvw_create_manager(NULL, "manager");
xvw_set_attributes(manager,
             XVW_WIDTH,  200,
             XVW_HEIGHT, 200,
                NULL);

/*
 *  create a yellow button in the middle of the manager;
    *  install a callback which will display the notifier.
 */
button = xvw_create_button(manager, "button");
xvw_set_attributes(button,
             XVW_LABEL,            "Press Me",
             XVW_BACKGROUND_COLOR, "yellow",
             XVW_LEFT_OF,           NULL,
             XVW_RIGHT_OF,          NULL,
             XVW_ABOVE,             NULL,
             XVW_BELOW,             NULL,
             NULL);

/* create a quit button, just to be fancy.  */
quit = xvw_create_button(manager, "quit");
   xvw_set_attributes(quit,
                      XVW_LABEL,            "quit",
                      XVW_LEFT_OF,           NULL,
                      XVW_BELOW,             NULL,
                      NULL);
/*
 *  Create the notify window object. Since the parent is NULL, a
 *  toplevel window will be created and the notifywindow placed
 *  inside.  Give the notify window a title and a message.
    *
    *  The XVW_NOTIFYWINDOW_VISIBLE attribute is used when an application
    *  will be repeatedly using a notify window;  you don't want to
    *  recreate it every time (since that's slow), so you just create it
    *  the first time, and then set the XVW_NOTIFYWINDOW_VISIBLE to
    *  FALSE after it's done, and back to TRUE again when putting
    *  up a new message.
 */
notifywindow = xvw_create_notifywindow(NULL, "notify");
xvw_set_attributes(notifywindow,
                XVW_NOTIFYWINDOW_VISIBLE, FALSE,
                XVW_NOTIFYWINDOW_TITLE,   "Please Wait. . .",
                NULL);

/*
 * add callback to button to display notifywindow.  pass
    * notifywindow as clientdata so we can use it inside the callback.
```

```
        */
       xvw_add_callback(button, XVW_BUTTON_SELECT,
                  button_cb, notifywindow);

       /*
        * add callback to button to quit.  pass notifywindow as clientdata
           * so we can destroy it inside the callback.
        */
       xvw_add_callback(quit, XVW_BUTTON_SELECT,
                  quit_cb, notifywindow);

       /* display and run the program.  */
       xvf_run_form();
}

/*
 *  callback to display notify window
 */
static void button_cb(
    xvobject object,
    kaddr    client_data,
    kaddr    call_data)
{
       int      indx;
       static   int count = 0;
       xvobject notifywindow = (xvobject) client_data;

       static  char *messages[] = {
                   "Initializing program...",
                   "the weather is here - wish you were beautiful",
                   "all in a day's work",
                   "isn't this fun",
                   "notifywindows are great"};

       indx = count % (knumber(messages));
       xvw_set_attributes(notifywindow,
                       XVW_NOTIFYWINDOW_MESSAGE, messages[indx],
                       XVW_NOTIFYWINDOW_VISIBLE, TRUE,
                       NULL);
       sleep(10);

       xvw_set_attribute(notifywindow, XVW_NOTIFYWINDOW_VISIBLE, FALSE);

       count++;
}

/*
 *  callback to quit
 */
static void quit_cb(
    xvobject object,
    kaddr    client_data,
    kaddr    call_data)
{
          xvobject notifywindow = (xvobject) client_data;
       xvw_destroy(notifywindow);
       kexit(KEXIT_SUCCESS);
}
```

N. The Outputfile Object

Figure 13: The OutputFile GUI object provides an output file window in which the user may enter an output filename.

N.1. xvw_create_outputfile() — *create a outputfile GUI object*

Synopsis

```
xvobject xvw_create_outputfile(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: parent of the outputfile widget; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The outputfile object on success, NULL on failure

Description

The outputfile GUI object allows the user to enter an output file. It features a text box in which the user may type in the filename; the filename is registered when the user hits <cr>. The label of the outputfile GUI object is really a button, which may be used to display the file browser. The file browser provides an alternate way for output file selection.

The outputfile GUI object consists of a manager object with three children: a button object, a text object, and a pixmap object.

A callback can be installed on the outputfile object, which will be fired when the user enters a new filename.

N.2. Attributes of the OutputFile Object

Summary of OutputFile Attributes	
Attribute	**Description**
XVW_OUTPUTFILE_BROWSER_OBJECT	This *read-only* attribute allows you to obtain the browser object which is popped up when the user clicks on the button of the outputfile object.
XVW_OUTPUTFILE_BUTTON_OBJECT	This *read-only* attribute allows you to obtain the button object component of the the outputfile object.

XVW_OUTPUTFILE_CALLBACK	If desired, *xvw_add_callback()* may be used to install a callback on the outputfile object which will be fired when the user enters a new filename, either by typing it in and pressing <cr>, or by using the browser. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(outputfileobj, XVW_OUTPUTFILE_CALLBACK, outputfile_cb, client_data);`
	Note that the current filename of the output file object will be passed to the callback in the *call_data*. The value must be cast to a string before use, as in:
	`char *filename = *((char **) call_data);`
	Alternatively, the filename may be obtained with XVW_OUTPUT-FILE_FILENAME.

XVW_OUTPUTFILE_CRLABEL_OBJECT	This *read-only* attribute allows you to obtain the pixmap object component of the the outputfile object that indicates a "live" selection.
XVW_OUTPUTFILE_DISPLAY_BUTTON	This attribute controls whether the browser button should be displayed (mapped) or not. If not then the text object
XVW_OUTPUTFILE_FILENAME	The filename which is currently displayed in the text object. This attribute can be used to initialize the filename to be displayed in the outputfile object, or to acquire the filename that has been entered by the user in the text object.
XVW_OUTPUTFILE_LABEL	This is the text that will appear in the label object component of the outputfile object. Provide text appropriate as a title of the outputfile object.
XVW_OUTPUTFILE_TEXT_OBJECT	This *read-only* attribute allows you to obtain the text object component of the the outputfile object.

Descriptions of OutputFile Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_OUTPUTFILE_BROWSER_OBJECT (N/A)	xvobject	NULL	The browser object (read-only).
XVW_OUTPUTFILE_BUTTON_OBJECT (N/A)	xvobject	NULL	The button object (read-only).

Descriptions of OutputFile Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_OUTPUTFILE_CALLBACK (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_OUTPUTFILE_CRLABEL_OBJECT (N/A)	xvobject	NULL	The pixmap object (read-only).
XVW_OUTPUTFILE_DISPLAY_BUTTON (outputfileDisplayButton)	int	TRUE	TRUE/FALSE
XVW_OUTPUTFILE_FILENAME (N/A)	char *	NULL	any valid output file name
XVW_OUTPUTFILE_LABEL (N/A)	char *	NULL	any printable text
XVW_OUTPUTFILE_TEXT_OBJECT (N/A)	xvobject	NULL	The text object (read-only).

N.3. Complete Resource Set of the OutputFile Object

The complete resource set for the outputfile object includes:

1. The outputfile object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3,"The kwidgets Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object."

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

N.4. Example using the OutputFile Object

An example program using the outputfile object can be found in `$DESIGN/examples/xvobjects/outputfile/example.c`.

```
#include <design.h>

/*
 *  This example creates a simple outputfile GUI object, which may
 *  be used for allowing the user to enter a output file.  The user may
 *  enter a value in the text parameter box, or click on the label button,
 *  which will bring up the file browser from which a file may be picked.
 *
 *  A callback is installed on the output file object so that when the
```

```
 *  user changes the value of the output file, the current
 *  filename is printed to the tty.
 *
 *  Note that the output file object should *not* be created directly in an
 *  xvroutine, as use of the OutputFile (-O) UIS line in the *.form file is
 *  both easier to use and a more standard use of the VisiQuest system.
 *  However, the outputfile object is provided for use with hybrid xvroutines,
 *  (such as this example) which do not use a "formalized" GUI as defined
 *  in a *.form file.
 */
static void outputfile_cb PROTO((xvobject, kaddr, kaddr));

main(
   int  argc,
   char *argv[])
{
     xvobject manager;
     xvobject object;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /* create a manager to be a parent for the outputfile object */
     manager = xvw_create_manager(NULL, "parent");
     xvw_set_attributes(manager,
                  XVW_WIDTH,       300,
                     XVW_HEIGHT,      100,
                  NULL);

     /*
         *  Create the outputfile object.  give it a label.
         *  tack it horizontally to the parent so that it spans the
         *  width of the manager backplane.  center it in the middle of
         *  the parent.
      */
     object = xvw_create_outputfile(manager, "outputfile");
     xvw_set_attributes(object,
          XVW_OUTPUTFILE_LABEL,     "Output File",
          XVW_OUTPUTFILE_FILENAME,  "lulu",
          XVW_TACK_EDGE,             KMANAGER_TACK_HORIZ,
          XVW_ABOVE,           NULL,
          XVW_BELOW,           NULL,
          NULL);

     xvw_add_callback(object, XVW_OUTPUTFILE_CALLBACK,
                outputfile_cb, NULL);

     /* display & run the program */
     xvf_run_form();
}

/*
```

```
 *  the callback for the outputfile will be fired when the user changes the
 *  value of the output file & hits <cr>, or when they choose a file using
 *  the browser.  this callback simply prints the current filename.
 */
static void outputfile_cb(
    xvobject object,
    kaddr client_data,
    kaddr call_data)
{
     char *filename;

     xvw_get_attribute(object, XVW_OUTPUTFILE_FILENAME, &filename);
     kfprintf(kstderr, "Filename = %s\n", filename);
}
```

O. The TextDisplay Object

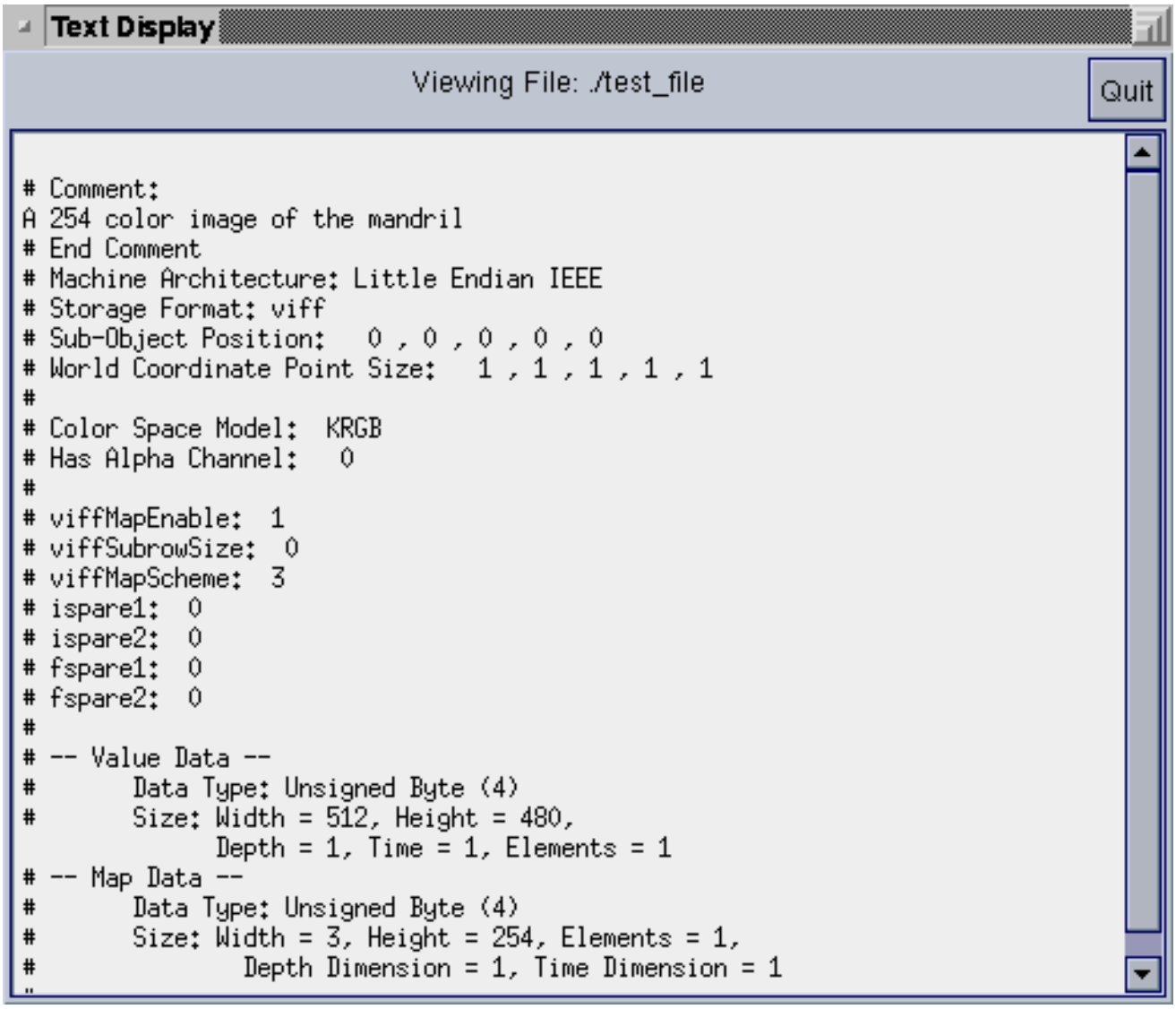


Figure 14: The upper portion of the textdisplay object, displaying the output of the man page for *xvw_create_textdisplay()*. Note that the output of the manpage contains the control characters produced by groff, and the textdisplay object correctly interprets the control characters as bold.

O.1. xvw_create_textdisplay() — *create a textdisplay object*

Synopsis

```
xvobject xvw_create_textdisplay(
   xvobject parent,
```

```
char     *name)
```

Input Arguments

`parent`
parent of the textdisplay object; NULL will cause a default toplevel to be created automatically

`name`
a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The textdisplay object on success, NULL on failure

Description

The textdisplay object supports display of text files.

For files that were output from *roff* interpreters, the textdisplay object will format the text as specified by the control characters produced by the *roff* interpreter. In other words, the textdisplay object cannot format files with *roff* commands still in them; rather, it supports display of files that were output by a *roff* interpreter such as *nroff* or *groff*, after such a program was already run on the file with the *roff* commands in it.

Of course, the textdisplay object also supports display of plain ascii text files.

The textdisplay object supports specification of five different fonts, the roman font that is used with "normal" text, as well as bold, italic, and helvetica fonts for emphasis and a symbol font for use with equations.

The textdisplay object can also be used to support hypertext-type functionality. The textdisplay object is capable of keeping a special list of words that might appear in the text; such "special" words will appear in a different color from the rest of the text. A callback can be installed on the textdisplay object which will be fired when the user clicks on a word that is part of the textdisplay list of "special" words. Thus, the application can use this capability to display a new text file determined by the word selected by the user.

O.2. Attributes of the TextDisplay Object

Summary of TextDisplay Attributes	
Attribute	**Description**
`XVW_TEXTDISPLAY_ADDTEXT`	This *action attribute* adds text to the text display. The text is appended to the existing list of text and is appropriately displayed after being compiled. The text passed via the *action attribute* is duplicated.

Summary of TextDisplay Attributes	
Attribute	**Description**
XVW_TEXTDISPLAY_ASCII	The XFontStruct structure defining the Ascii font to be used with "normal" text appearing in the text display object. The XFontStruct value for setting this attribute can be obtained with *XLoadQueryFont()*; a list of XFontStruct candidates may be obtained with *XListFontsWithInfo()*. For more information on these Xlib routines, see Section 6.2 of *The Xlib Programming Manual* by Adrian Nye. Note that this attribute is mutually exclusive with XVW_TEXTDISPLAY_ASCII_FONTNAME; use one or the other, not both.
XVW_TEXTDISPLAY_ASCII_FONTNAME	This attribute specifies the name of the Ascii font to be used with "normal" text appearing in the text display object.
XVW_TEXTDISPLAY_CALLBACK	If desired, *xvw_add_callback()* may be used to install a callback on the text display object that will be fired when the user clicks on a particular word that appears in the text being displayed. When calling *xvw_add_callback()*, pass this attribute directly, as in `xvw_add_callback(textdisplayobj, XVW_TEXTDISPLAY_CALLBACK, textdisplay_cb, client_data);` When this callback attribute is used, either the XVW_TEXTDISPLAY_WORDCOLOR or the XVW_TEXTDISPLAY_WORDPIXEL attribute should be used to set the color of the word to something other than the rest of the text. This will indicate to the user that they can click on the word to make something happen (what will happen is, of course, determined by the callback). In addition, it is also good to set either the XVW_TEXTDISPLAY_HIGHLIGHTCOLOR or the XVW_TEXTDISPLAY_HIGHLIGHTPIXEL attribute to specify the color in which the word will appear when it is actually clicked on. Note that the word that the user clicked on will be passed as a string in the *call_data*; to access the string inside the callback, it must first cast to a string, as in: `char *word = (char *) calldata`
XVW_TEXTDISPLAY_CLEARTEXT	This *action attribute* deletes all text from the text display. All text will be deleted from the text display list, which will result in the text display defaulting to it's initial state.
XVW_TEXTDISPLAY_FILE	This is the open stream to the file to be displayed in the text display object. Note that this attribute is mutually exclusive with XVW_TEXTDISPLAY_FILENAME; use one or the other, not both.
XVW_TEXTDISPLAY_FILENAME	This is the name of the file to be displayed in the display object.
XVW_TEXTDISPLAY_GETTEXT	This *action attribute* gets text from the text display. The text passed back is not duplicated and should not be changed.
XVW_TEXTDISPLAY_INDENT	The number of pixels to indent into the text display object before printing text.

Summary of TextDisplay Attributes	
Attribute	**Description**
XVW_TEXTDISPLAY_ROFF	If the file to appear in the text display object has *roff* formatting commands, setting this attribute to TRUE will cause those formatting commands to be interpreted and the text formatted as specified. If FALSE, the file to be displayed is considered to be a plain ascii text file, and the text is displayed exactly as it appears in the file.
XVW_TEXTDISPLAY_ROMAN	The XFontStruct structure defining the Roman font to be used with "normal" text appearing in the text display object. The XFontStruct value for setting this attribute can be obtained with *XLoadQueryFont()*; a list of XFontStruct candidates may be obtained with *XListFontsWithInfo()*. For more information on these Xlib routines, see Section 6.2 of *The Xlib Programming Manual* by Adrian Nye. Note that this attribute is mutually exclusive with XVW_TEXTDISPLAY_ROMAN_FONTNAME; use one or the other, not both.
XVW_TEXTDISPLAY_ROMAN_FONTNAME	This attribute specifies the name of the Roman font to be used with "normal" text appearing in the text display object.

Descriptions of TextDisplay Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_TEXTDISPLAY_ADDTEXT (N/A)	char *	N/A	Any multiline text which is displayable
XVW_TEXTDISPLAY_ASCII (N/A)	XFontStruct	see description	see description
XVW_TEXTDISPLAY_ASCII_FONTNAME (textdisplayAsciiFontname)	char *	see description	see description
XVW_TEXTDISPLAY_CALLBACK (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_TEXTDISPLAY_CLEARTEXT (N/A)	int	N/A	TRUE
XVW_TEXTDISPLAY_FILE (N/A)	kfile	NULL	A *kfile* pointer returned by *kfopen()* or another VisiQuest open data transport routine
XVW_TEXTDISPLAY_FILENAME (textdisplayFilename)	char *	NULL	valid name of input file
XVW_TEXTDISPLAY_GETTEXT (N/A)	char *	N/A	N/A

Descriptions of TextDisplay Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_TEXTDISPLAY_INDENT (textdisplayIndent)	int	10	values > 0
XVW_TEXTDISPLAY_ROFF (textdisplayRoff)	int	TRUE	TRUE/FALSE
XVW_TEXTDISPLAY_ROMAN (N/A)	XFontStruct	see description	see description
XVW_TEXTDISPLAY_ROMAN_FONTNAME (textdisplayRomanFontname)	char *	see description	see description

O.3. Complete Resource Set of the Textdisplay Object

The complete resource set for the textdisplay object includes:

1. The textdisplay object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3,"The kwidgets Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object."

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

O.4. Example using the Textdisplay Object

An example program using the textdisplay object can be found in `$DESIGN/examples/xvobjects/textdisplay/example.c`.

```
#include <design.h>

/*
 *  This example shows how the textdisplay object is used to display
 *  a text file.  If the file is formatted with "roff" commands, those
 *  commands will be interpreted, and the file formatted as specified.
 */

static void quit_cb  PROTO((xvobject, kaddr, kaddr));

xvobject text;
xvobject back;
xvobject label;

void main(
   int  argc,
   char **argv)
{
```

```
xvobject quit;
floatwidth;
char *title;
char    *filename = "./test_file";
kfile   *file;
kobject current;

   /* initialize VisiQuest program */
   khoros_initialize(argc, argv, "DESIGN");

/* initialize the xvwidgets lib */
if (!xvw_initialize(XVW_MENUS_XVFORMS))
{
   kerror(NULL, "example", "Cannot open display");
   kexit(KEXIT_FAILURE);
}


/* create the manager backplane object */
back = xvw_create_manager(NULL, "Text Display");
xvw_set_attributes(back,
          XVW_PREFERRED_WIDTH,      250,            /* set width     */
          XVW_PREFERRED_HEIGHT,     400,            /* set height    */
       NULL);


/* create the label object */
title = kstring_cat("Viewing File: ", filename, NULL);
width = (float) (kstrlen(title)) +1.0;
label = xvw_create_label(back, "label");
xvw_set_attributes(label,
     XVW_LABEL,     title,
     XVW_CHAR_WIDTH,     20.0,
     XVW_BELOW,     NULL,
     XVW_RIGHT_OF,       NULL,
     XVW_LEFT_OF,        NULL,
     NULL);

/* create the quit button */
quit = xvw_create_button(back, "quit");
xvw_set_attributes(quit,
     XVW_LABEL,     "Quit",        /* button label   */
     XVW_LEFT_OF,        NULL,     /* upper R corner */
     XVW_BELOW,     NULL,     /* upper R corner */
     XVW_CHAR_HEIGHT,    1.0,      /* set height     */
     NULL);


file = kfopen(filename, "r");
   if (file == NULL)
   {
       kerror(NULL, "get_image_info",
              "Can't open file %s", filename);
       return;
   }

/* create the textdisplay object; it will display the test file */
text = xvw_create_textdisplay(back, "test_display");
xvw_set_attributes(text,
```

```
            XVW_BELOW,          quit,
               XVW_TEXTDISPLAY_FILE,     file,        /* set filename  */
            XVW_TACK_EDGE,           KMANAGER_TACK_ALL,
               NULL);

     kfclose(file);

     /* add event handler to quit when they hit the quit button */
     xvw_add_callback(quit, XVW_BUTTON_SELECT, quit_cb, NULL);

     /* quit the program if they use the window manager to delete window */
     xvw_add_protocol(back, "WM_DELETE_WINDOW",
                (void (*)(xvobject, kaddr))quit_cb, NULL);

     /* display & run program */
     xvf_run_form();
}

static void quit_cb(
   xvobject object,
   kaddr  client_data,
   kaddr  call_data)
{

     xvw_remove_protocol(back, "WM_DELETE_WINDOW",
                  (void (*)(xvobject, kaddr))quit_cb, NULL);
        xvw_unmap(back);
        xvw_destroy(back);

     kexit(KEXIT_SUCCESS);

}
```

P. The TextInput Object

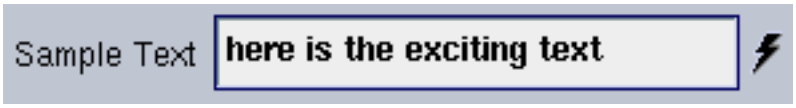


Figure 15: The TextInput GUI object provides an text input window in which the user may enter an text string.

P.1. xvw_create_textinput() — *create a textinput object*

Synopsis

```
xvobject xvw_create_textinput(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: parent of the textinput object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The textinput object on success, NULL on failure

Description

The textinput GUI object allows the user to enter a string of arbitrary length. It features a text box in which the user may enter the string; the string is registered when the user hits <cr>.

A callback can be installed on the textinput object, which will be fired when the user enters a new string.

P.2. Attributes of the TextInput Object

Summary of TextInput Attributes	
Attribute	**Description**

XVW_TEXTINPUT_CALLBACK	If desired, *xvw_add_callback()* may be used to install a callback on the textinput object which will be fired when the user enters new text and presses <cr>. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(textinputobj, XVW_TEXTINPUT_CALLBACK, textinput_cb, client_data);`
	Note that the current text appearing in the textinput object will be passed to the callback in the *call_data*. The value must be cast to a string before use, as in:
	`char *string = *((char **) call_data);`
	Alternatively, the filename may be obtained with XVW_TEXTINPUT_TEXT.

XVW_TEXTINPUT_CRLABEL_OBJECT	This *read-only* attribute allows you to obtain the pixmap object component of the the textinput object that indicates a "live" selection.
XVW_TEXTINPUT_LABEL	This is the label that appears on the label object component of the textinput object.
XVW_TEXTINPUT_LABEL_OBJECT	This *read-only* attribute allows you to obtain the label object component of the the textinput object.
XVW_TEXTINPUT_TEXT	This is the text that appears in the textinput object.
XVW_TEXTINPUT_TEXT_OBJECT	This *read-only* attribute allows you to obtain the text object component of the the textinput object.

Descriptions of TextInput Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**

XVW_TEXTINPUT_CALLBACK (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form:
			
			`void callback_function`
			`xvobject object,`
			`kaddr client_data,`
			`kaddr call_data)`

XVW_TEXTINPUT_CRLABEL_OBJECT (N/A)	xvobject	NULL	The pixmap object (read-only).
XVW_TEXTINPUT_LABEL (N/A)	char *		
XVW_TEXTINPUT_LABEL_OBJECT (N/A)	xvobject	NULL	The label object (read-only).

Descriptions of TextInput Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_TEXTINPUT_TEXT (N/A)	char *	NULL	any printable text
XVW_TEXTINPUT_TEXT_OBJECT (N/A)	xvobject	NULL	The text object (read-only).

P.3. Complete Resource Set of the TextInput Object

The complete resource set for the textinput object includes:

1. The textinput object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3,"The kwidgets Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object."

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

.section 2 "Example using the TextInput Object" An example program using the textinput object can be found in `$DESIGN/examples/xvobjects/textinput/example.c`.

```
#include <design.h>

/*
 *  This example creates a simple textinput GUI object, which may
 *  be used for allowing the user to enter text.
 *
 *  A callback is installed on the textinput object so that when the
 *  user changes the text, the current text is printed to the tty.
 *
 *  Note that the textinput object should *not* be created directly in an
 *  xvroutine, as use of the String (-s) UIS line in the *.form file is
 *  both easier to use and a more standard use of the VisiQuest system.
 *  However, the textinput object is provided for use with hybrid xvroutines,
 *  (such as this example) which do not use a "formalized" GUI as defined
 *  in a *.form file.
 */
static void textinput_cb PROTO((xvobject, kaddr, kaddr));

main(
   int  argc,
   char *argv[])
{
     xvobject manager;
     xvobject object;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");
```

```
    /* initialize the xvwidgets lib */
    if (!xvw_initialize(XVW_MENUS_XVFORMS))
    {
       kerror(NULL, "main", "unable to open display");
       kexit(KEXIT_FAILURE);
    }

    /* create a manager backplane */
    manager = xvw_create_manager(NULL, "parent");
    xvw_set_attributes(manager,
                XVW_WIDTH,       300,
                   XVW_HEIGHT,      100,
                NULL);

    /*
        *  Create the textinput object.  give it a label.
        *  tack it horizontally to the parent so that it spans the
        *  width of the manager backplane.  center it in the middle of
        *  the parent.
     */
    object = xvw_create_textinput(manager, "textinput");
    xvw_set_attributes(object,
         XVW_TEXTINPUT_LABEL,     "Sample Text",
         XVW_TEXTINPUT_TEXT,      "here is the exciting text",
         XVW_TACK_EDGE,           KMANAGER_TACK_HORIZ,
         XVW_ABOVE,         NULL,
         XVW_BELOW,         NULL,
         NULL);
    xvw_add_callback(object, XVW_TEXTINPUT_CALLBACK, textinput_cb, NULL);

    /* display & run the program */
    xvf_run_form();
}

/*
 *  the callback for the textinput will be fired when the user changes the
 *  value of the text & hits <cr>. this callback simply prints the text.
 */
static void textinput_cb(
    xvobject object,
    kaddr client_data,
    kaddr call_data)
{
     char *text;

     xvw_get_attribute(object, XVW_TEXTINPUT_TEXT, &text);
     kfprintf(kstderr, "Text reads as follows:\n%s\n", text);
}
```

Q. The Warn Object

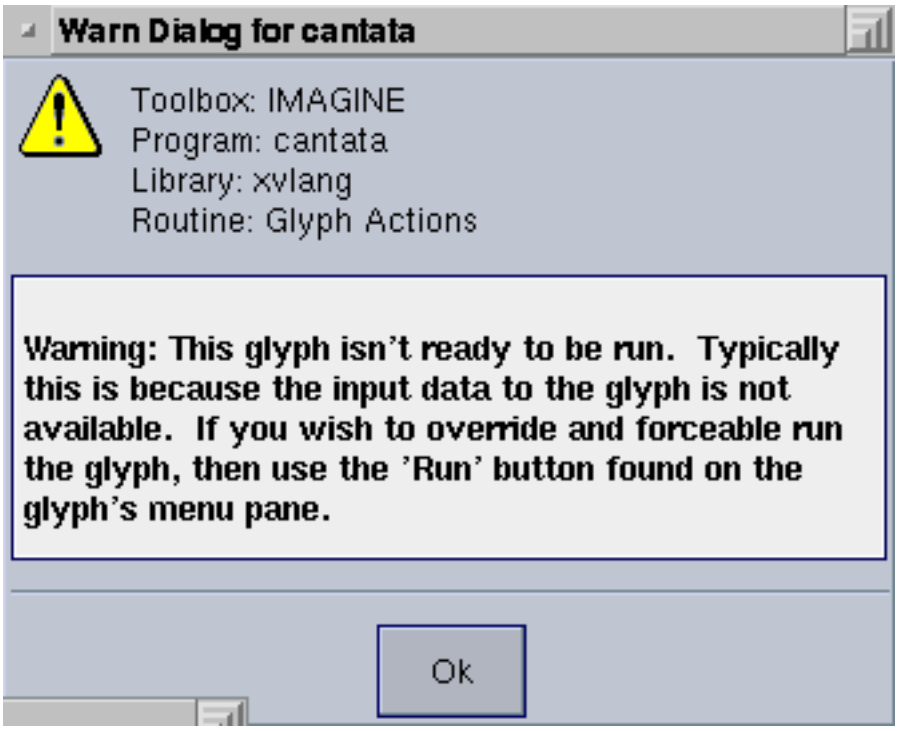


Figure 16: The warn object is most often used indirectly, through *kwarn()*, to print warning messages.

Q.1. xvw_create_warn() — *create a warning object*

Synopsis

```
xvobject xvw_create_warn(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: parent of the warn object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the object (for use in app-defaults files, etc)

Returns

The warn widget on success, NULL on failure

Description

Creates a warning object. The warning object is a pop-up window that displays a warning message. A warn icon on the upper left hand side of the warning object draws the attention of the user; a single button on the upper right hand side of the warning object allows the user to acknowledge the warning.

After the user clicks on the acknowledgment button, the warning object is destroyed.

A callback can be installed on the warning object, which will be fired when the user clicks on the acknowledgment button.

Q.2. Attributes of the Warn Object

Summary of Warn Attributes

Attribute	Description
XVW_WARN_BUTTON_LABEL	The label for the acknowledgment button.
XVW_WARN_BUTTON_OBJECT	This *read-only* attribute allows you to obtain the the button object component of the warn object (the button used to allow the user to acknowledge the warning message).

XVW_WARN_CALLBACK	If desired, *xvw_add_callback()* may be used to install a callback on the warn object which will be fired when the user clicks on the acknowledgement button. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(warnobj, XVW_WARN_CALLBACK,`
	`warn_cb, client_data);`

XVW_WARN_LABEL	The label for the warn object.
XVW_WARN_LABEL_OBJECT	This *read-only* attribute allows you to obtain the label object component of the warn object (the object used to display the label).
XVW_WARN_MESSAGE	The error message to be displayed in the error object.
XVW_WARN_PIXMAP	This is the pixmap that appears to the upper left of the warn object. Candidates for the value of this attribute may be created with the use of *XCreatePixmap()*; see *The Xlib Reference Manual* by O'Reilly and Associates. Note that this attribute is mutually exclusive with XVW_WARN_PIXMAPFILE; specify one or the other, not both.
XVW_WARN_PIXMAPFILE	This is the file defining the pixmap that appears at the upper left of the warn object.
XVW_WARN_TEXT_OBJECT	This *read-only* attribute allows you to obtain the the text object component of the warn object (the text displaying the warning message).

Descriptions of Warn Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_WARN_BUTTON_LABEL (N/A)	char *	"Ok"	any printable text
XVW_WARN_BUTTON_OBJECT (N/A)	xvobject	NULL	The button object (read-only).
XVW_WARN_CALLBACK (N/A)	void (*call-back_rou-tine)(xvob-ject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_WARN_LABEL (N/A)	char *	NULL	any printable text
XVW_WARN_LABEL_OBJECT (N/A)	xvobject	NULL	The label object (read-only).
XVW_WARN_MESSAGE (N/A)	char *	NULL	any printable text
XVW_WARN_PIXMAP (N/A)	Pixmap	The "caution" pixmap.	Valid Pixmap structure
XVW_WARN_PIXMAPFILE (warnPixmapfile)	char *	The caution.xpm file in the *xvob-jects/misc/pixmaps* directory, which defines the "cau-tion" pixmap.	The full path to a valid xpm or xbm file, defining the desired pixmap. Note that the path may contain $TOOLBOX.
XVW_WARN_TEXT_OBJECT (N/A)	xvobject	NULL	The text object (read-only).

Q.3. Complete Resource Set of the Warn Object

The complete resource set for the warn object includes:

1. The warn object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3,"The kwidgets Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object."

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

Q.4. Example using the Warn Object

An example program using the warn object can be found in `$DESIGN/examples/xvobjects/warn/example.c`.

```
#include <design.h>

/*
 *  This example creates a warn object to display a warninig message.
 *
 *  IMPORTANT NOTE: in general, the warn object should *not* be created
 *                  directly;  use of kwarn() is the conventional way to
 *                  create messages in VisiQuest, so that there is standard
 *                  formatting enforced.  this example is really only for
 *                  academic purposes.
 */
void main(
   int  argc,
   char *argv[])
{
     xvobject warn;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /*
      *  Create the warn object. Note that since the parent is NULL, a
      *  toplevel window will be created and the warn object placed
      *  inside.   Set the label, the warning message, the button label.
      */
     warn = xvw_create_warn(NULL, "Warning Message");
     xvw_set_attributes(warn,
          XVW_WARN_LABEL,   "Beeeep!  Beeeep!  Beeep!",
          XVW_WARN_MESSAGE, "You better watch out...",
          NULL);

     /* display & run the program */
     xvf_run_form();
}
```

This page left intentionally blank

Table of Contents

Chapter 4

The Graphics Attributes

Chapter 4 - The Graphics Attributes

The visual objects in the *xvisual*, *xvplot*, and *xvannotate* libraries inherit the attributes of the graphics class, called *graphics attributes*. The graphics attributes are only listed once, in this section. The objects which are subclassed from the graphics class and inherit the *graphics attributes* are listed below.

- The area object
- The 2D axis object
- The circle object
- The date object
- The ellipse object
- The image object
- The indicator object
- The line object
- The marker object
- The 2D plot object
- The 3D plot object
- The polyline object
- The rectangle object
- The string object
- The stringvalue object
- The text string object
- The timer object
- The zoom object

You can set or get the graphics attributes on any of the visual objects listed above without error. However, simply because a visual object has inherited an attribute from its Graphics superclass does not necessarily mean that it will use the attribute. It is the nature of the visual object itself that will dictate whether or not a particular graphics attribute applies.

In general, common sense will be a sufficient guide to determine which attributes apply to which visual objects. For example, the circle object is as a closed shape; thus, it can be expected that setting `XVW_GRAPHICS_FILLED` to TRUE on the circle object will cause the circle object to be filled. The line object, on the other hand, is not a filled shape; accordingly, it can be expected that setting `XVW_GRAPHICS_FILLED` to TRUE on the line object will have no effect.

Some graphics attributes are applicable only to a certain group of visual objects that share a particular characteristic, such as a marker component or *subpart*. Such attributes are defined by the graphics class rather than by the particular objects in question because other objects which also need the attribute would not otherwise inherit it. For example, it is desirable to be able to set the marker type with `XVW_GRAPHICS_MARKERTYPE` on a 2D plot object for use when it is displaying a scatterplot or a linemarker plot. Since the 2D plot object is not subclassed from the marker object, this "marker-specific" attribute would not be available to the 2D plot object if it was an attribute of the marker itself. By making the marker type an attribute of the graphics class, the `XVW_GRAPHICS_MARKERTYPE` attribute can be set not only on the marker object, but also on the 2D plot object, the 3D plot object, and the indicator object, all of which also use markers.

Graphics attributes can be loosely categorized into two groups: "appearance" attributes and "world view" attributes. Appearance attributes provide control over the appearance of the visual object, while world view attributes control how they are displayed with respect to their "world."

A. Appearance Attributes

A few of the graphics attributes can be generally applied to control the appearance of most visual objects. For example, most visual objects can be filled with the foreground color by setting XVW_GRAPHICS_FILLED to TRUE. Visual objects that have line components can have their line type and line width controlled with XVW_GRAPHICS_LINETYPE and XVW_GRAPHICS_LINEWIDTH. Those visual objects that have one or more marker components can have the marker scale and marker type set with XVW_GRAPHICS_MARKERSCALE and XVW_GRAPHICS_MARKERTYPE.

Summary of Appearance Attributes

Attribute	Description
XVW_GRAPHICS_FILLED	When set to true, this attribute indicates that the object should be filled with the foreground color; objects are not filled by default.
XVW_GRAPHICS_LINETYPE	Visual objects involving a line may specify the line type.
XVW_GRAPHICS_LINEWIDTH	Visual objects involving a line may specify the line width.
XVW_GRAPHICS_MARKERSCALE	Visual objects involving one or more markers may specify the size of marker(s).
XVW_GRAPHICS_MARKERTYPE	Visual objects involving one or more markers may specify the marker type to be used by the marker(s). Choices include an arc, a bow tie, a box, a caret, a circle, a cross, a dagger, a diamond, a dot, a hexagon, a point, a pixel, a square, a triangle, an "X", or a "V".

Descriptions of Appearance Attributes

Attribute (Resource Name)	Type	Default	Legal Values
XVW_GRAPHICS_FILLED (graphicsFilled)	int	FALSE	TRUE/FALSE
XVW_GRAPHICS_LINETYPE (graphicsLinetype)	int	KLINE_SOLID	KLINE_SOLID KLINE_DOTTED KLINE_DOT_DASH KLINE_SHORT_DASH KLINE_LONG_DASH KLINE_ODD_DASH KLINE_GRID_DOTTED

Descriptions of Appearance Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_GRAPHICS_LINEWIDTH (graphicsLinewidth)	int	KLINE_EXTRA_FINE	KLINE_EXTRA_FINE KLINE_FINE KLINE_MEDIUM_FINE KLINE_MEDIUM KLINE_MEDIUM_WIDE KLINE_WIDE KLINE_EXTRA_WIDE
XVW_GRAPHICS_MARKERSCALE (graphicsMarkerScale)	integer	1	value >= 1
XVW_GRAPHICS_MARKERTYPE (graphicsMarkertype)	int	KMARKER_SQUARE	KMARKER_ARC KMARKER_BOW_TIE KMARKER_BOX KMARKER_CARET KMARKER_CIRCLE KMARKER_CROSS KMARKER_DAGGER KMARKER_DIAMOND KMARKER_DOT KMARKER_HEXAGON KMARKER_POINT KMARKER_PIXEL KMARKER_SQUARE KMARKER_TRIANGLE KMARKER_X KMARKER_V

B. World View Attributes

The *world view* of a visual object refers collectively to the current settings of the viewport maximums and minimums, the world coordinate maximums and minimums, the perspective, the axis mode, and the symmetry. The following sections explain how to use the attributes that are related to various aspects of the world view.

B.1. World Coordinates

The size and position of a visual object *in space* is described by its *world coordinates*. The *xvgraphics* library uses a cartesian coordinate system, the bounds of which are specified using *world coordinate maximums and minimums*, or "wcmins" and "wcmaxes." The application can specify any floating-point values for the wcmins and wcmaxes; the XVW_GRAPHICS_WCMIN_{X,Y,Z} and XVW_GRAPHICS_WCMAX_{X,Y,Z} attributes are used *on the parent object* [1] to specify the values. Visual objects should have world coordinate locations that

[1] By default, the parent of a visual object specifies its wcmins and wcmaxes; the exception to this is when the visual object is "attached" to itself or a sibling.

place them within the bounds set by the wcmins and wcmaxes. Unless a visual object is located within these bounds, it will not be displayed.

The attributes that are used to specify the world coordinate size and location of a particular visual object are defined by that object. For example, the rectangle object's world coordinates are specified with the XVW_RECTANGLE_X and XVW_RECTANGLE_Y attributes, while the circle object's world coordinates are specified with the XVW_CIRCLE_XCENTER and XVW_CIRCLE_YCENTER attributes. See the section on the visual object of interest for a listing and explanation of the attributes that are used to set its world coordinates.

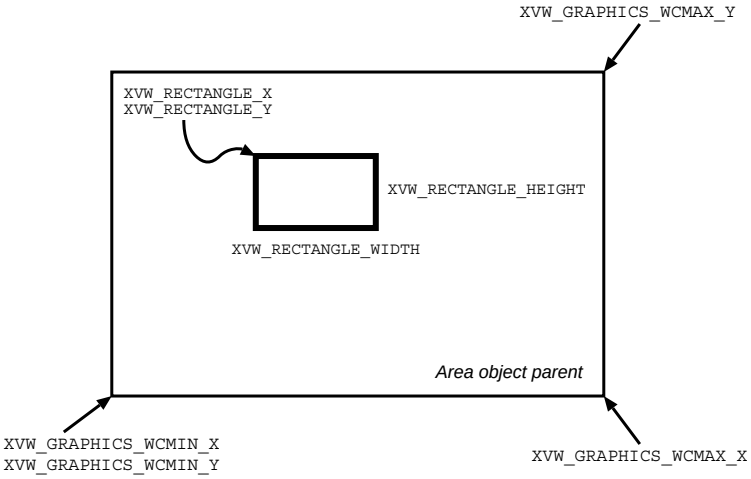


Figure 1: A rectangle annotation is positioned and sized within its area object parent using world coordinates. The world-coordinate range is specified by the wcmin and wcmax of the parent area object; the world-coordinate origin is located in the lower, left-hand corner.

The concept of world coordinates contrasts with that of *device coordinates*. The literal position, in pixels, of a visual object within its parent is defined by its device coordinates, the bounds of which are defined by the physical size of its parent. Sizing of visual objects with device coordinates is done using the XVW_WIDTH and XVW_HEIGHT attributes; positioning of visual objects with device coordinates is achieved using the XVW_XPOSITION and XVW_YPOSITION attributes; see section B.1 of Chapter 2, "The xvwidgets Library" for more information on these attributes.

It is important to understand that when device coordinate specifications are used to place a visual object, the device coordinates of the object *always* refer to the *upper, left-hand corner of the bounding box surrounding the object*. In contrast, the world coordinates of an object sometimes refers to the center of the object; read the explanation of the world coordinate attributes for the object in question to discover if this is the case. For example, the marker object, the circle object, and the ellipse object are three examples of visual objects that have world coordinate locations referring to the *center* of the object, while the device coordinate locations refer to the upper, left-hand corner of their bounding boxes.

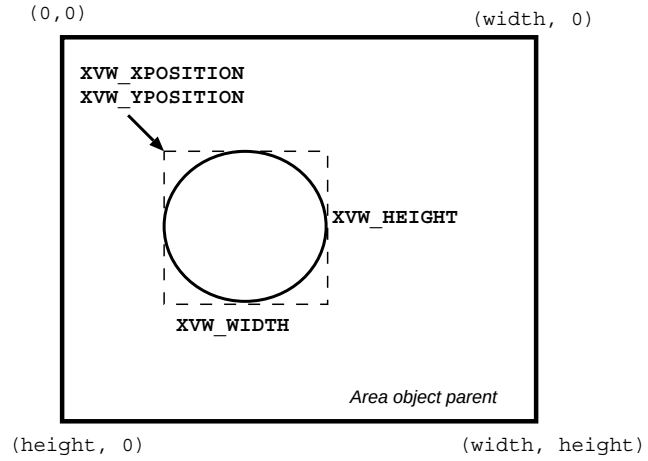


Figure 2: A circle annotation is positioned and sized within its area object parent using device coordinates. The device coordinate range is specified by the actual width and height of the parent area object; the device coordinate origin is located in the upper, left-hand corner.

Visual objects may be sized and positioned using either world coordinates or device coordinates. Of course, the actual display of visual objects is always done using device coordinates. Internally, world coordinate specifications are mapped into device coordinate specifications before visual objects are placed on the screen.

The use of world coordinates rather than device coordinates for sizing and placement of visual objects has important advantages. Since world coordinates are not dependent on the physical dimensions of the parent object, it is not necessary to know the size of the parent in order to place visual objects within it. Furthermore, the parent can be resized by the user without disturbing the size or location of the visual objects within it. Most importantly, since world coordinates can be set to any floating point values, they can be tailored to "make sense" within the context of any application.

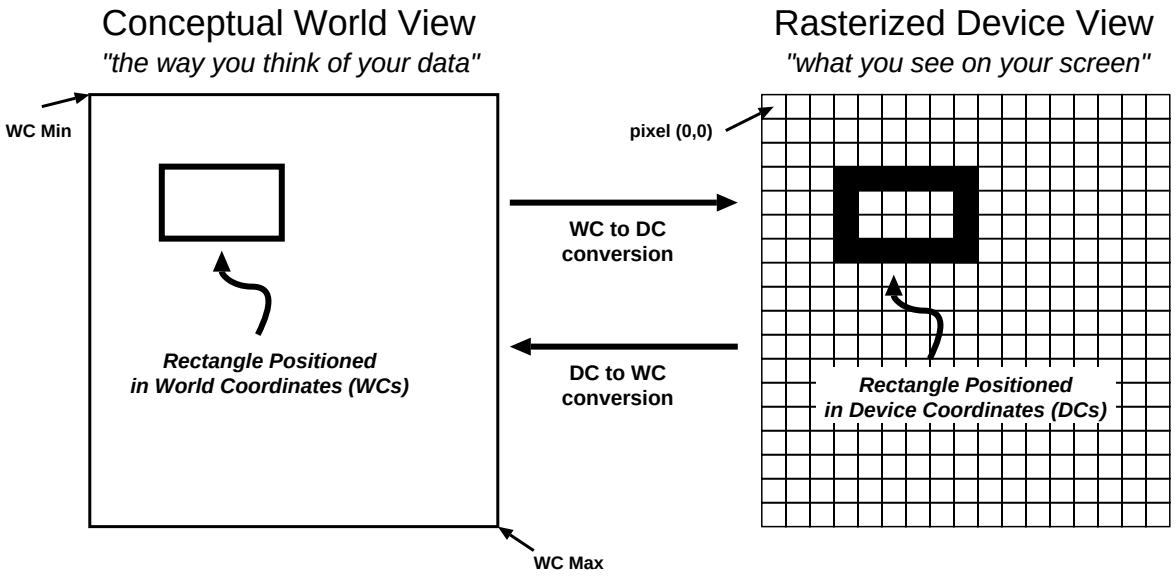


Figure 3: When world coordinates are used to specify the location of a visual object, the world coordinates are internally converted into device coordinates in preparation for actual display on the screen. While any visual object may be placed directly using device coordinates, world coordinates are often easier as well as more flexible to use than device coordinates.

For consistency, either world coordinates or device coordinates (not both) should be used with a particular instance of a visual object. For example, a circle might be placed using the world coordinate `XVW_CIRCLE_XCENTER` and `XVW_CIRCLE_YCENTER`, and sized using `XVW_CIRCLE_RADIUS`. Alternatively, it might be placed using the device coordinate `XVW_XPOSITION` and `XVW_YPOSITION`, and sized using `XVW_WIDTH` and `XVW_HEIGHT`. Mixing the use of world coordinate settings with device coordinate settings on the same instance of a particular visual object is not recommended, however. At best, the specifications will be confusing, while at worst, unexpected and incorrect results may occur.

By default, the X and Y wcmins are 0.0 and the X and Y wcmaxes are 1.0 [2] and the origin is considered to be in the lower, left-hand corner for all visual objects except image objects (a special case that will be discussed separately).

[2] World coordinate bounds are also from 0.0 to 1.0 in the Z direction for the 3D plot object.

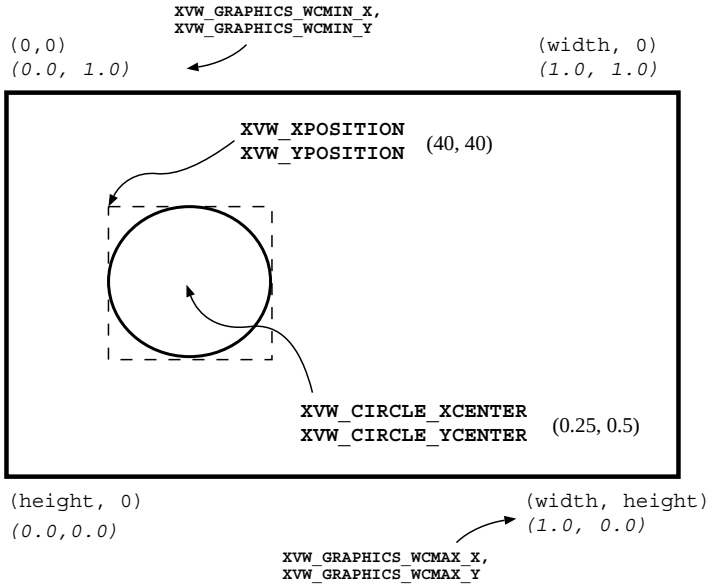


Figure 4: By default, the world coordinate range of an area object is (<0,0>, <1,1>). A circle annotation might be placed within an area object parent using a world coordinate specification of its center at (0.25, 0.25); alternatively, it might be placed with a device coordinate specification of the upper left hand corner of its bounding box at (40,40).

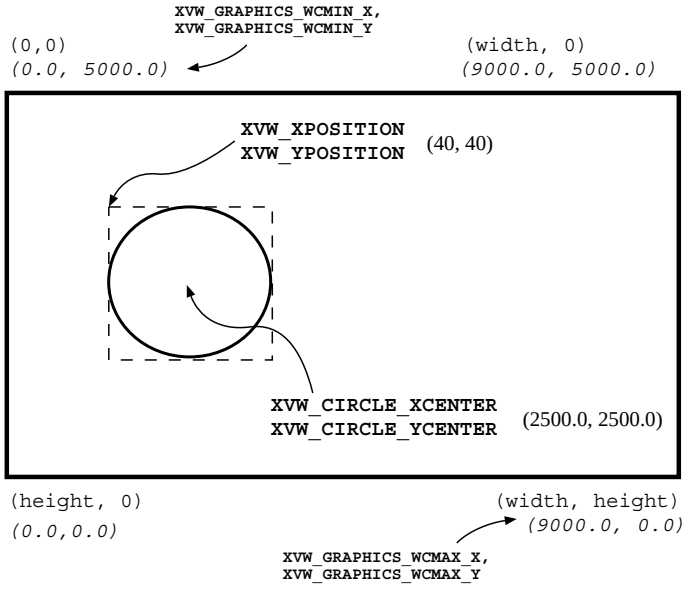


Figure 5: In this example, the wcmins and wcmaxes of the area object have been increased to (<0,9000>, <0,5000>). The device coordinate location of the circle annotation is the same as in the previous figure; however, the world coordinate location is now relative to the modified world coordinate range of the area object.

The image object presents a special case; its world coordinates are treated differently than other visual objects. Unless otherwise specified, the device coordinate values of an image object are the same as its world coordinate values. Like other visual objects, the wcmins of an image object are 0.0; however, the wcmaxes of the image object are automatically set according to its width and height instead of being set to 1.0 by default. Furthermore, the origin of an image object is in the upper left hand corner rather than in the lower left hand corner. This makes sizing and placement of visual objects (specifically annotations) within an image object consistent with the orientation of the image itself.

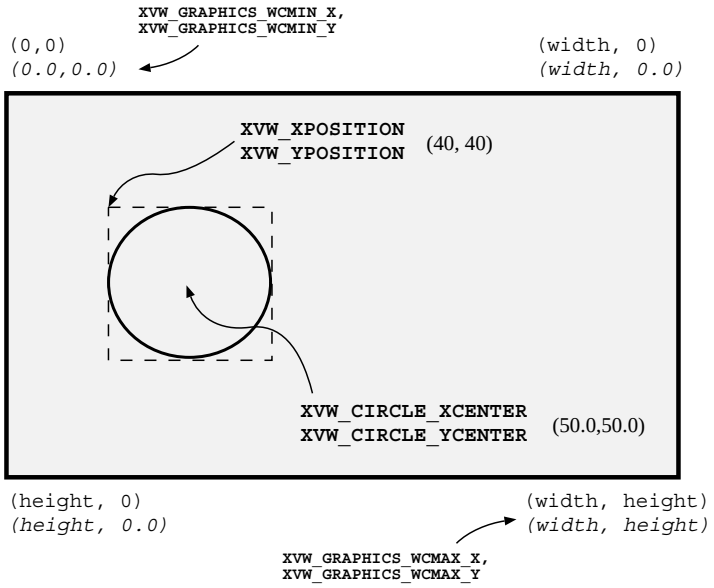


Figure 6: The image object

When visual objects such as annotations are created with an image object as their parent, world coordinates should *always* be used to size and place them. Since the world coordinate values of an image object are the same as the device coordinate values, any reason for preferring the use of device coordinates will not apply. The key issue, however, is support of large images. Suppose an image object is used to display an image that is too large to fit in the image window, and that a pan icon is needed to allow the user to control the portion of the image that appears in the image window. Suppose further that one or more visual objects (such as annotations) are created as children of the image object, so that they are displayed on top of the image. Only when world coordinates are used to place the annotations will they be able to maintain their relative position with respect to the image as the user pans about the image.

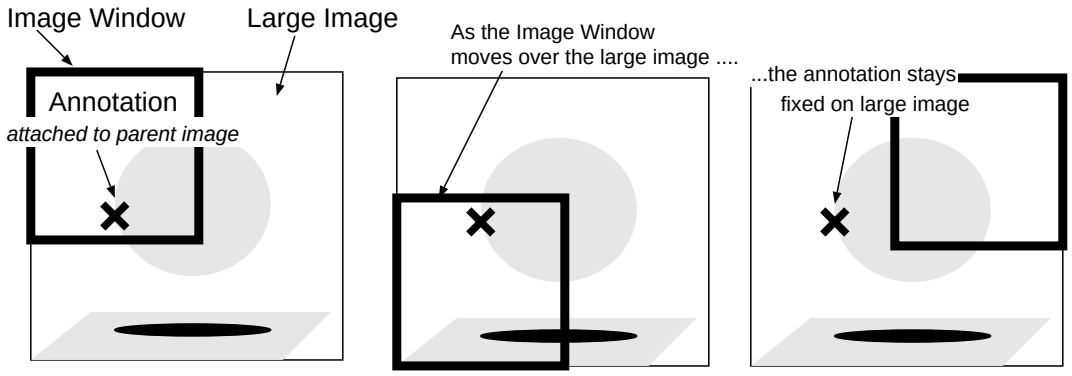


Figure 7: World coordinates must be used to place annotations on a large image, so that when the user pans about the image, the annotations will be able to maintain their correct positions on the image.

Summary of Attributes that Control World Coordinates

Attribute	Description
XVW_GRAPHICS_RESET_WORLD	This action attribute will cause the object to search all its children for the overall maximum and minimum, and will adjust the global maximum and minimum accordingly, so that all visual objects within the object are shown. Note that it is used *only* with the *area* object.
XVW_GRAPHICS_WCMAX_X	This attribute specifies the maximum value of the world coordinate range in the X direction. Note that this attribute should *only* be set on the visual object that is serving as the "controlling" visual object (see Section C.4, "Attaching the World View").
XVW_GRAPHICS_WCMAX_Y	his attribute specifies the maximum value of the world coordinate range in the Y direction. Note that this attribute should *only* be set on the visual object that is serving as the "controlling" visual object (see Section C.4, "Attaching the World View").
XVW_GRAPHICS_WCMAX_Z	This attribute specifies the maximum value of the world coordinate range in the Z direction. Note that this attribute should *only* be set on the visual object that is serving as the "controlling" visual object (see Section C.4, "Attaching the World View"). Note that this attribute only applies to 3D objects.
XVW_GRAPHICS_WCMIN_X	This attribute specifies the minimum value of the world coordinate range in the X direction. Note that this attribute should *only* be set on the visual object that is serving as the "controlling" visual object (see Section C.4, "Attaching the World View").
XVW_GRAPHICS_WCMIN_Y	This attribute specifies the minimum value of the world coordinate range in the Y direction. Note that this attribute should *only* be set on the visual object that is serving as the "controlling" visual object (see Section C.4, "Attaching the World View").

Summary of Attributes that Control World Coordinates	
Attribute	**Description**
XVW_GRAPHICS_WCMIN_Z	This attribute specifies the minimum value of the world coordinate range in the Z direction. Note that this attribute should *only* be set on the visual object that is serving as the "controlling" visual object (see Section C.4, "Attaching the World View"). Note that this attribute only applies to 3D objects.

Descriptions of Attributes that Control World Coordinates			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_GRAPHICS_RESET_WORLD (N/A)	int	N/A (action attribute)	TRUE/FALSE
XVW_GRAPHICS_WCMAX_X (N/A)	double	1.0	value > XVW_GRAPHICS_WCMIN_X
XVW_GRAPHICS_WCMAX_Y (N/A)	double	1.0	value > XVW_GRAPHICS_WCMIN_Y
XVW_GRAPHICS_WCMAX_Z (N/A)	double	1.0	value > XVW_GRAPHICS_WCMIN_Z
XVW_GRAPHICS_WCMIN_X (N/A)	double	0.0	value < XVW_GRAPHICS_WCMAX_X
XVW_GRAPHICS_WCMIN_Y (N/A)	double	0.0	value < XVW_GRAPHICS_WCMAX_Y
XVW_GRAPHICS_WCMIN_Z (N/A)	double	0.0	value < XVW_GRAPHICS_WCMAX_X

B.2. Viewport Coordinates

The viewport minimums and maximums of a visual object describe the size and location of the region in which children of the object may be displayed. Legal values for the viewport minimums and minimums range 0.0 to 1.0, where (0.0, 0.0) is the lower, left-hand corner of the object, and (1.0, 1.0) is the upper, right-hand corner of the object. Viewports produce a "buffer" of space around the region in which child objects are displayed, and are generally used to improve the general appearance of a visual display.

The entire world-coordinate extent of the object must be isolated within the bounds of the viewport. To achieve this, the wcmins and wcmaxes of the object are offset so that their locations match the locations of the viewport minimums and maximums. Because the viewport applies only to the placement of children within a visual object, setting the viewport will only have an effect when used with objects that lay out children. In particular, the area object is the object most frequently given a viewport; it is very rare that viewport settings are made on any other visual object.

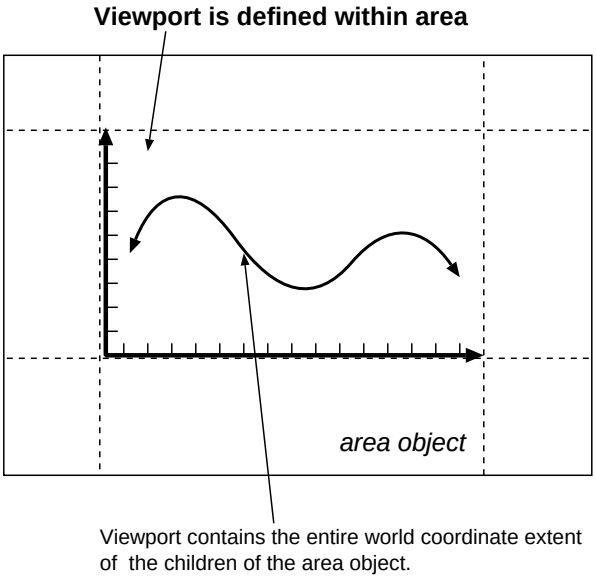


Figure 8: The wcmins and wcmaxes of the area object are offset so that they are within the bounds of the viewport.

For example, suppose a 2D visual object is created inside an area object where the world coordinate minimum is (2.0, 2.0) and maximum is (4.0, 4.0). If area object had its viewport minimum set to (0.25, 0.25) and its viewport maximum set to (1.0, 1.0), the region containing the object would be offset to the upper right.

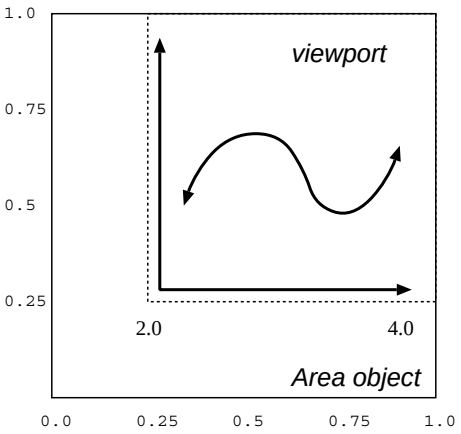


Figure 9: An axis object and a plot object, created as children of an area object having its viewport set to (<0.25,1.0>, <0.25,1.0>).

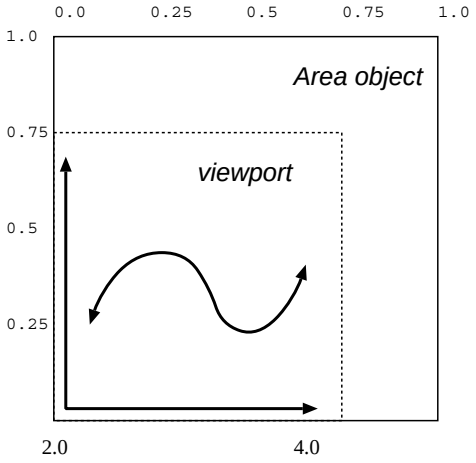


Figure 10: If the same area object has its viewport set to (<0.0,0.0>, <0.75,0.75>), the portion of the area object in which the axis and the plot is displayed will be moved to the lower, left corner.

By default, the viewport is set to (<0.0,0.0>, <1.0,1.0>), which is the entire object; thus, by default, the viewport will have no effect. The most common use for a viewport is on an area object which is to contain a 2D axis object and one or more 2D plot objects. In this case, a viewport is recommended to provide some space between the 2D axis and the edges of the area object. A recommended setting for the viewport of the area object is (<0.1,0.1>, <0.9,0.9>) if the 2D axis will not be labelled, or (<0.2,0.9>, <0.2,0.9>) if the 2D axis will have labels for the X and Y axes.

Summary of Attributes that Control the Viewport

Attribute	Description
XVW_GRAPHICS_VIEWPORT_MAX_X	This attribute specifies the maximum viewport value in the X direction. Note that this attribute should *only* be set on the visual object that is serving as the "controlling" visual object (see Section C.4, "Attaching the World View").
XVW_GRAPHICS_VIEWPORT_MAX_Y	This attribute specifies the maximum viewport value in the Y direction. Note that this attribute should *only* be set on the visual object that is serving as the "controlling" visual object (see Section C.4, "Attaching the World View").
XVW_GRAPHICS_VIEWPORT_MAX_Z	This attribute specifies the maximum viewport value in the Z direction. Note that this attribute should *only* be set on the visual object that is serving as the "controlling" visual object (see Section C.4, "Attaching the World View"). Note that this attribute is only applicable to 3D objects.
XVW_GRAPHICS_VIEWPORT_MIN_X	This attribute specifies the minimum viewport value in the X direction. Note that this attribute should *only* be set on the visual object that is serving as the "controlling" visual object (see Section C.4, "Attaching the World View").

Summary of Attributes that Control the Viewport	
Attribute	**Description**
XVW_GRAPHICS_VIEWPORT_MIN_Y	This attribute specifies the minimum viewport value in the Y direction. Note that this attribute should *only* be set on the visual object that is serving as the "controlling" visual object (see Section C.4, "Attaching the World View").
XVW_GRAPHICS_VIEWPORT_MIN_Z	This attribute specifies the minimum viewport value in the Z direction. Note that this attribute should *only* be set on the visual object that is serving as the "controlling" visual object (see Section C.4, "Attaching the World View"). Note that this attribute is only applicable to 3D objects.

Descriptions of Attributes that Control the Viewport			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_GRAPHICS_VIEWPORT_MAX_X (graphicsViewportMaxX)	double	1.0	value > XVW_GRAPHICS_VIEWPORT_MIN_X
XVW_GRAPHICS_VIEWPORT_MAX_Y (graphicsViewportMaxY)	double	1.0	value > XVW_GRAPHICS_VIEWPORT_MIN_Y
XVW_GRAPHICS_VIEWPORT_MAX_Z (graphicsViewportMaxZ)	double	1.0	value > XVW_GRAPHICS_VIEWPORT_MIN_Z
XVW_GRAPHICS_VIEWPORT_MIN_X (graphicsViewportMinX)	double	0.0	value < XVW_GRAPHICS_VIEWPORT_MAX_X
XVW_GRAPHICS_VIEWPORT_MIN_Y (graphicsViewportMinY)	double	0.0	value < XVW_GRAPHICS_VIEWPORT_MAX_Y
XVW_GRAPHICS_VIEWPORT_MIN_Z (graphicsViewportMinZ)	double	0.0	value < XVW_GRAPHICS_VIEWPORT_MAX_Z

B.3. Perspective

These attributes controls the viewing perspective orientation.

Summary of Attributes that Control 3D Perspective	
Attribute	**Description**
XVW_GRAPHICS_ALPHA	Rotates the camera (the eye) about the X axis.
XVW_GRAPHICS_EYE	The eye distance moves the camera (the *eye*) either closer to or farther away from the plot. Setting the eye distance to 0 would be as if the eye was touching the object.
XVW_GRAPHICS_GAMMA	Rotates the camera (the eye) about the Y axis.

Summary of Attributes that Control 3D Perspective	
Attribute	**Description**
XVW_GRAPHICS_PROJECTION	This attribute specifies the projection type. Supported settings include: KGRAPHICS_PERSPECTIVE- a projection in which parallel lines not parallel to the projection plane converge to a vanishing point. KGRAPHICS_ORTHOGRAPHIC- a parallel projection in which the direction of projection and the normal to the projection plane are identical. KGRAPHICS_CAVALIER- an oblique projection in which the direction of projection makes a 45 degree angle with the direction of the projection plane; as a result, the projection of a line perpendicular to the projection plane is the same length as the line itself. KGRAPHICS_CABINET- an oblique projection in which the direction of projection makes an angle of arccot (1/2) with the projection plane so that lines perpendicular to the projection plane project at half their actual length; this projection is considered more realistic than the cavalier projection.
XVW_GRAPHICS_THETA	Rotates the camera (the eye) about the Z axis.
XVW_GRAPHICS_VIEWDISTANCE	The distance of the viewport from the object.

Descriptions of Attributes that Control 3D Perspective			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_GRAPHICS_ALPHA (graphicsAlpha)	double	300.0	0.0 - 360.0 (alpha is given in degrees)
XVW_GRAPHICS_EYE (graphicsEye)	double	1.0	value > 0.0
XVW_GRAPHICS_GAMMA (graphicsGamma)	double	0.0	0.0 - 360.0 (gamma is given in degrees)
XVW_GRAPHICS_PROJECTION (graphicsProjection)	int	KGRAPHICS_PERSPECTIVE	KGRAPHICS_PERSPECTIVE KGRAPHICS_ORTHOGRAPHIC KGRAPHICS_CAVALIER KGRAPHICS_CABINET
XVW_GRAPHICS_THETA (graphicsTheta)	double	70.0	0.0 - 360.0 (theta is given in degrees)
XVW_GRAPHICS_VIEWDISTANCE (graphicsViewdistance)	double	6.0	any values within the world coordinate system

B.4. The Axis Mode

The axis mode may be linear, natural log, or log base 10. Note that natural log is not supported yet.

Summary of Attributes That Control Axis Mode

Attribute	Description
XVW_GRAPHICS_MODE_X	Whether the X Axis is marked as linear or log 10 scale.
XVW_GRAPHICS_MODE_Y	Whether the Y Axis is marked as linear or log 10 scale.
XVW_GRAPHICS_MODE_Z	Whether the Z Axis is marked as linear or log 10 scale.
XVW_GRAPHICS_PROPORTIONAL	When this attribute is set to KGRAPHICS_NONPROP, the scale in each axis direction is determined by the world coordinate minimums and maximums of the data. Thus, if the X values range from 0 to 1, and the Y values range from 1 to 100, and the physical length of the X and Y axes are the same, then the scale in the X direction (1) is much smaller than the scale in the Y direction (100). In contrast, when this attribute is set to KGRAPHICS_PROP_WINDOWED or KGRAPHICS_PROP_NONWINDOWED, it causes the scale in each axis direction to be the same. When a visual object is proportional, this means that the minimum and maximum values across being displayed as X and Y are found before the object is displayed. Then the minimum and maximum are set on each axis so that the scale is the same. If KGRAPHICS_PROP_WINDOWED is specified, then the center of the data is determined along each axis and the minimum and maximum values are set such that the data is in the center of the world coordinate space. If KGRAPHICS_PROP_NONWINDOWED is specified, the range of numbers covered, i.e, the minimum and maximum values are the same, by the X and Y axes are the same. The "sense of proportion" conveyed by the object is correct, but details of the object may be lost if the range spanned by X varies greatly from that spanned by Y. In some cases, proportional display not be a good method of display. For example, suppose a 2D plot object has data displayed as X ranges from 0 to 255, but the data displayed as Y ranges from 0 to 1. Obviously, the plot will not be very informative, as all the points will be bunched against the X axis. In cases like this, non-proportional display may be more useful.

Descriptions of Attributes That Control Axis Mode

Attribute (Resource Name)	Type	Default	Legal Values
XVW_GRAPHICS_MODE_X (graphicsModeX)	int	KGRAPHICS_LINEAR	KGRAPHICS_LINEAR KGRAPHICS_LOG10

Descriptions of Attributes That Control Axis Mode			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_GRAPHICS_MODE_Y (graphicsModeY)	int	KGRAPHICS_LINEAR	KGRAPHICS_LINEAR KGRAPHICS_LOG10
XVW_GRAPHICS_MODE_Z (graphicsModeZ)	int	KGRAPHICS_LINEAR	KGRAPHICS_LINEAR KGRAPHICS_LOG10
XVW_GRAPHICS_PROPORTIONAL (graphicsProportional)	int	KGRAPHICS_NONPROP	KGRAPHICS_NONPROP KGRAPHICS_PROP_WINDOWED KGRAPHICS_PROP_NONWINDOWED

B.5. Symmetry

When a visual object is symmetrical, the world coordinate maximums and minimums of the object will be equal in magnitude and opposite in sign. For example, suppose the world coordinate minimum of a visual object in the X direction is (-1.0) and the world coordinate maximum of the visual object in the X direction is (0.5). When symmetry is turned on, the world coordinate maximum will be changed to (1.0), since (-1.0) is larger in magnitude than (0.5).

Summary of Attributes that Control Symmetry	
Attribute	**Description**
XVW_GRAPHICS_SYMMETRIC_X	When TRUE, this attribute specifies that world coordinates are symmetric in the X direction.
XVW_GRAPHICS_SYMMETRIC_Y	When TRUE, this attribute specifies that world coordinates are symmetric in the Y direction.
XVW_GRAPHICS_SYMMETRIC_Z	When TRUE, this attribute specifies that world coordinates are symmetric in the Z direction.

Descriptions of Attributes that Control Symmetry			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_GRAPHICS_SYMMETRIC_X (graphicsSymmetricX)	int	FALSE	TRUE/FALSE
XVW_GRAPHICS_SYMMETRIC_Y (graphicsSymmetricY)	int	FALSE	TRUE/FALSE
XVW_GRAPHICS_SYMMETRIC_Z (graphicsSymmetricZ)	int	FALSE	TRUE/FALSE

C. Clipping

Clipping causes any portion of an object that is outside the viewport to be not shown, or "clipped".

Summary of Attributes That Control Clipping	
Attribute	**Description**
XVW_GRAPHICS_CLIPPING	When TRUE, this attribute turns clipping on; FALSE causes clipping to be turned off. When clipping is on, any parts of a visual object that have X and Y world coordinates that lie outside the world coordinate maximum and minimum values specified for X and Y will not be drawn. So, for example, an application might implement the capability to "zoom in" and "zoom out" on a plot simply by changing the world coordinate maximums and minimums while clipping is on.
XVW_GRAPHICS_DEPTH_CLIPPING	When TRUE, this attribute allows a 3D visual object to be clipped with respect to depth. When depth clipping is on, any portions of a visual object that have Z world coordinates outside the world coordinate maximum and minimum values for Z will not be drawn.

Descriptions of Attributes That Control Clipping			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_GRAPHICS_CLIPPING (graphicsClipping)	int	TRUE	TRUE/FALSE
XVW_GRAPHICS_DEPTH_CLIPPING (graphicsDepthClipping)	int	TRUE	TRUE/FALSE

D. Attaching The World View

Recall that the "world view" of a visual object is the current settings of its viewport maximums and minimums, its world coordinate maximums and minimums, its perspective, axis mode, and symmetry.

Frequently, it is necessary that a number of visual objects share the same world view. In this event, it is necessary to specify *which* visual object will have the "authority" to "impose" its world view on the other visual objects. This is done by *attaching* visual objects to one another with the use of the XVW_GRAPHICS_ATTACH attribute. When a call such as the following is made:

```
xvw_set_attribute(objectA, XVW_GRAPHICS_ATTACH, objectB);
```

it is said that objectA is *attached* to objectB. That is, objectA will get its view of the world from objectB; objectB will be "in control," and will impose its world view on objectA.

An object may be attached to its parent, to itself, or to a sibling. Its placement depends on how it is attached, as does its behavior when any aspect of the world view is modified.

Summary of Attach Attribute	
Attribute	**Description**
XVW_GRAPHICS_ATTACH	This attribute indicates the object from which another object will acquire its world coordinate view. See section C.4, "Attaching the World View", for a complete explanation of the XVW_GRAPHICS_ATTACH attribute.

Description of Attach Attribute			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_GRAPHICS_ATTACH (N/A)	xvobject	NULL	The object itself, the object's parent, or NULL

D.1. Attaching An Object To Its Parent

By default, a visual object is attached to its parent. Thus, the parent will dictate the world view of its children. The following code:

```
xvw_set_attribute(object, XVW_GRAPHICS_ATTACH, parent);
```

is redundant, since the object is already attached to its parent by default.

In a very simple case, consider a marker annotation that is created as a child of an area object. By default, the marker annotation will inherit the world view of the area object. Also by default, the world coordinate minimums and maximums of the area object are 0.0 to 1.0; thus, the marker annotation will also have world coordinates ranging from 0.0 to 1.0. If the marker is given a world coordinate placement of (0.1, 0.1), it will appear in the upper left hand corner of the area object; if the marker is given a world coordinate placement of (0.9, 0.9), it will appear in the lower right hand corner of the area object; if the marker is given a world coordinate placement of (2.0, 5.0), this will be outside its world view, and it will not appear on the area object at all. If the world coordinate maximums and minimums of the area object are changed, the new values will be passed on to the marker.

In this way, an area object might be used to contain a number of other visual objects, all of which will be contained in a consistent, coordinated world view. The world view attributes should be set only on the area object; because each of the visual objects is attached to its parent by default, they will all "inherit" the updated world view. For example, suppose an area object is the parent of several circle objects. The following code:

```
coord min, max;
min.x = -2.5; max.x = 6.3;
min.y = -2.5; max.y = 7.8;
xvw_set_attribute(area, XVW_GRAPHICS_WCMIN, min);
xvw_set_attribute(area, XVW_GRAPHICS_WCMAX, max);
```

will change the world coordinate range of all the circles from the default (<0.0, 1.0>, <0.0, 1.0>) to a new range of (<-2.5, 2.5>, <6.3, 7.8>).

Another valuable use of this feature is with respect to the annotation of a displayed image. Suppose an image object is created to display an image, and a number of annotations such as markers, circles, rectangles, and so

on are to appear at various locations about the image. The marker, circle, and rectangle visual objects may be created with the image object as the parent, and placed as desired about the image using world coordinate placement attributes. Because the annotations will inherit the world view of the image, the world coordinate range of the markers will be (<0, 0>, <w, h>), where w and h are the width and height of the image, and the origin is in the upper left hand corner of the image. [3]

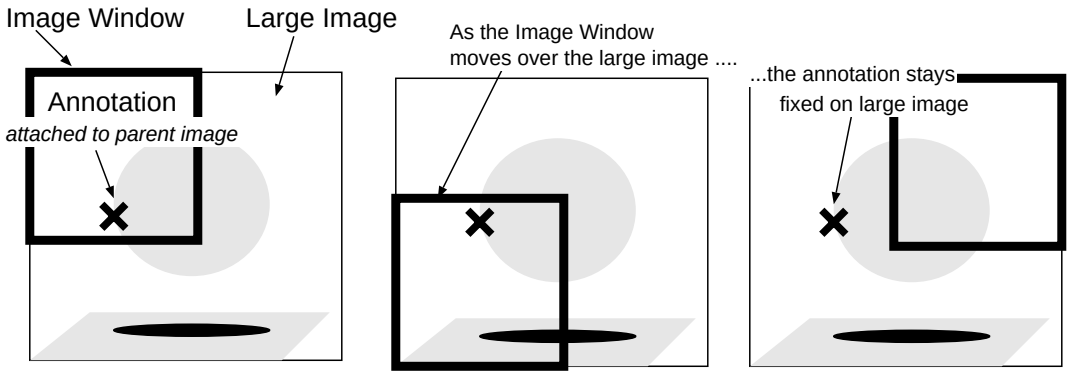


Figure 11: When an annotation is attached to a parent image, and a pan icon is used to pan about a large image, the annotation will maintain its correct location with respect to the world coordinates of the image.

D.2. Attaching An Object To Itself

Since an object is attached to its parent by default, changes that are made to its own world view will have no effect. If an object is to be able to control its own world view, it must be explicitly attached *to itself*, as in the following:

```
xvw_set_attribute(axis2d, XVW_GRAPHICS_ATTACH, axis2d);
```

Only when the axis object is attached to itself can it update itself correctly.

In contrast to the case described in the previous section, where an annotation is attached to its parent, suppose an annotation which is created as the child of an image object is explicitly attached to itself. By default, the world coordinate range of the annotation is (<0,0>, <1,1>), and the origin is at the lower left hand corner of the image.

[3] Note that the location of the origin differs between an area object and an image object; the origin of an area object at the lower left, while the origin of an image is at the upper left.

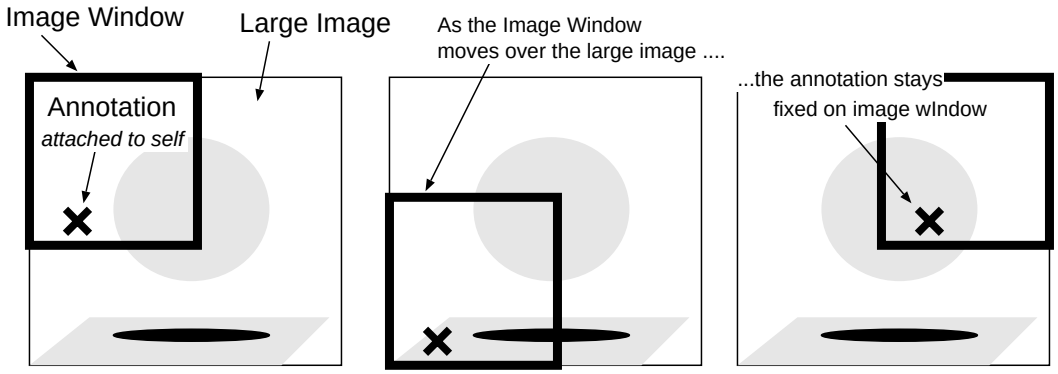


Figure 12: When an annotation is attached to itself rather than the parent image, and a pan icon is used to pan about a large image, the annotation will maintain its location with respect to its own world coordinates rather than those of the image, and therefore will appear at the same location in the image window, regardless of panning.

There are a number of cases, most commonly those associated with plotting, where it is necessary that an object be attached to itself so that it can determine its own world view. For example, consider the case of a 2D axis that is the child of an area object. If the user of the application changes the values of the viewport, world coordinate range, axis mode, perspective, or symmetry via the menuform of the 2D axis object, he will of course expect the displayed axes to update appropriately. However, this cannot happen unless the axis has control over its own world view.

D.3. Attaching An Object To A Sibling

In some cases, notably those involving both axes and plots, it is necessary for one child of a parent object to control the world view of one or more other children of the same parent object.

For example, suppose an application presents data in the form of three 2D plots labeled with an axis. Internally, this is implemented using an area object as the parent of a 2D axis object and three 2D plot objects. If the user of the application changes the world coordinate minimums via the menuform of the 2D axis object, he will not only expect the labels on the axis to change, but also that the three plots will update so as to present the data in a way that is consistent with the labels on the axis.

If the axis object is attached to itself, it will update itself correctly (see the previous section). However, in order to cause the 2D axis object to control the world view of the three plot objects, the three plot objects must be attached to the axis:

```
xvw_set_attribute(plot1, XVW_GRAPHICS_ATTACH, axis2d);
xvw_set_attribute(plot2, XVW_GRAPHICS_ATTACH, axis2d);
xvw_set_attribute(plot3, XVW_GRAPHICS_ATTACH, axis2d);
```

When this is done, any change to the world view performed via the 2D axis object will be "propagated" to the three plot objects, which will then update correctly and consistently with the axis.

Table of Contents

This page left intentionally blank

Chapter 5

Xvimage

Chapter 5 - Xvimage

A. Introduction

The *xvisual* library offers visual objects for the display of images, as well as visual objects for the display and manipulation of colormap information. The *xvisual* library also contains the *color class*, from which all visual objects in Visualization Services are subclassed. This chapter begins by giving an overview of the visual objects provided by the *xvisual* library. Next, it details the color attributes which are inherited by the visual objects in Visualization Services. Finally, the details on the various visual objects provided by the *xvisual* library are given.

A.1. Overview of Visual Objects Related To Imaging

With respect to imaging, the key visual object provided by the *xvimage* library is, of course, the image object. The imaging capabilities of the image object are sophisticated and comprehensive. The image object directly supports display of bit, byte, short, integer, float, and even complex images; no data type conversion is necessary. Full support for 24-bit displays is provided.

Zooming on images is provided by the *zoom* visual object; the *animation* visual object provides a quick, easy way to do animation of image sequences. Large images may have a *pan icon* visual object created to allow the user to change the portion of the image that appears in the image window, while any image may be made into an icon using the *image icon* visual object. Regular and irregular *regions of interest* may be interactively extracted from and inserted into the displayed image using an attribute of the *image* visual object created for that purpose. Pixel values of an image may be monitored with a *printpixel* visual object. The visual objects related to imaging include: the animate object, the image object, the imageicon object the panicon object, the position object, the printpixel object, and the zoom object.

Available Functions

- *xvw_create_animate()* - create a slide animation visual object
- *xvw_create_image()* - create an image object
- *xvw_create_imageicon()* - create a imageicon object
- *xvw_create_panicon()* - create a panicon object
- *xvw_create_position()* - create a position object
- *xvw_create_printpixel()* - create a printpixel xvobject
- *xvw_create_zoom()* - create a zoom object

A.2. Overview of Visual Objects Related To Colormap Manipulation

There are variety of methods for changing colormaps. While Color Services offers a variety of different predefined colormaps as well as a number of colormap operations that may be performed on the currently installed colormap, the *xvimage* library provides a variety of methods for interactive manipulation of colormaps to augment the noninteractive colormap algorithms defined by data services. A *colorcell* visual object may be used to display the color in which a pixel currently appears in the image; the colorcell may further be

used to change that color according to the needs of the application. The *pseudocolor* visual object operates allow you to change a range or discontinous set of values in the colormap to a desired color specified with RGB values; the colors in the map may be depicted using a set of colorcells, a palette, or a color wheel.

Values in the colormap surrounding the current location of the pointer may be printed in a display with the *printmapval* visual object; interactive thresholding and windowing are also supported by the *threshold* visual object. Values in the maps may be normalized according to the maximum displayable intensities of a screen, or with standard deviation normalizations; values displayed as red, green and blue may be specified by any of the columns of map data provided with an image; for more sophisticated needs, the values displayed as red, green, and blue may be determined using functions across any or all of the map columns. The visual objects related to colormap manipulation include: the colorcell object, the palette object, the printmapval object, the pseudo object, and the threshold object.

Available Functions

- *xvw_create_colorcell()* - create a colorcell xvobject
- *xvw_create_palette()* - create a palette object
- *xvw_create_printmapval()* - create a printmapval xvobject
- *xvw_create_pseudo()* - create a pseudo xvobject
- *xvw_create_threshold()* - create a threshold object

B. The Color Attributes

All visual objects in the *xvimage* library are subclassed from the color object. The attributes they inherit from the color class are referred to as *color attributes*. These attributes are only listed once, in this section; they are referenced by the sections associated with all visual objects with which they are associated. Note that visual objects in the *xvannotate* and *xvplot* libraries also inherit these color attributes.

Summary of Color Class Attributes	
Attribute	**Description**
XVW_COLOR_ALLOC_POLICY	A *visual* describes the characteristics of a virtual colormap that is created for use with an application on a particular screen. For a complete explanation of visuals, you are referred to Chapters 7.2, 7.3, and 7.4 of *The XLib Programming Manual*, by Adrian Nye, published by O'Reilly & Associates. Depending on your particular workstation, you may have a DirectColor, GreyScale, PseudoColor, StaticColor, StaticGrey, or TrueColor visual.
XVW_COLOR_BEGIN	Use this attribute to specify the beginning of the range of color indices to be allocated, if that range is to be limited to something other than the full number of indices in the colormap.
XVW_COLOR_BLUE_FUNCTION	This is the function defining the blue value of each pixel in the image. By default, the XVW_COLOR_BLUE_FUNCTION attribute is set to *M2*, or the third map column; thus, it produces the results that you would normally expect for an image with a colormap, where no function was being applied. See the explanation for XVW_COLOR_RED_FUNCTION for more details.
XVW_COLOR_BLUE_MAPCOL	This attribute controls which map column is used to specify the blue values of the pixels in the image; see XVW_COLOR_RED_MAPCOL for more details.
XVW_COLOR_CALLBACK	If desired, you may install a callback that will be fired when the colormap is changed. The colormap may be changed by an autocolor procedure or a colormap operation being applied, a pseudocoloring operation, a change in the map data of the data object being displayed, or any other modification of the colormap.
XVW_COLOR_CHANGE_BLUE_MAPCOL	This is an *action* attribute; i.e, it may only be used with *xvw_set_attribute(s)()*. Provide TRUE as the value. See XVW_COLOR_RED_MAPCOL for more details.
XVW_COLOR_CHANGE_GREEN_MAPCOL	This is an *action* attribute; i.e, it may only be used with *xvw_set_attribute(s)()*. Provide TRUE as the value. See XVW_COLOR_RED_MAPCOL for more details.
XVW_COLOR_CHANGE_RED_MAPCOL	This is an *action* attribute; i.e, it may only be used with *xvw_set_attribute(s)()*. Provide TRUE as the value. This attribute provides a convenient, interactive front end that an application may use to allow the user to set the XVW_COLOR_RED_MAPCOL attribute. It will query the number of map columns associated with the image, and present the user with a pop-up list of the map columns available. The user will be allowed to choose one of the map columns from the list; then, XVW_COLOR_RED_MAPCOL will be automatically set according to the choice made by the user.
XVW_COLOR_COLORFILE	The name of the file containing the colormap to be displayed. Note that this attribute is mutually exclusive with XVW_COLOR_COLOROBJ; set one or the other, not both.

Summary of Color Class Attributes	
Attribute	**Description**
XVW_COLOR_COLOROBJ	This is the data object containing the colormap to be displayed. Note that this attribute is mutually exclusive with XVW_COLOR_COLORFILE; set one or the other, not both.
XVW_COLOR_COLORS	This *read only* attribute can be used to obtain the XColor array representing the map data for the displayed data object. Red, green, and blue values are normalized between 0 and 65,536. Pixel values reflect those that were allocated in order to display the colormap.
XVW_COLOR_END	Use this attribute to specify the end of the range of color indices to be allocated, if that range is to be limited to something other than the full number of indices in the colormap.
XVW_COLOR_GREEN_FUNCTION	This is the function defining the green value of each pixel in the image. By default, the XVW_COLOR_GREEN_FUNCTION attribute is set to *M1*, or the second map column; thus, it produces the results that you would normally expect for an image with a colormap, where no function was being applied. See the explanation for XVW_COLOR_RED_FUNCTION for more details.
XVW_COLOR_GREEN_MAPCOL	This attribute controls which map column is used to specify the green values of the pixels in the image; see XVW_COLOR_RED_MAPCOL for more details.
XVW_COLOR_NORM_METHOD	When normalization is to be performed, this attribute defines the algorithm which is used in normalization. Normalization options allow different emphasis on how the data is displayed. KCOLOR_NORM_BESTGUESS tries to choose the correct normalization by looking at the data to be displayed. If the data is floating point and between 0 and 1 (inclusive), the normalization chooses a minimum of 0 and a maximum of 1 and scales accordingly. Otherwise, if all the data is between 0 and 255 (inclusive), normalization chooses a minimum of 0 and a maximum of 255. Note that global normalization will always be used in these cases. If neither of these conditions is met, then KCOLOR_NORM_MAXCOLORS will be used for the normalization. Using KCOLOR_NORM_NONE allows normalization to be turned off. This option assumes the data is already scaled between 0 and 65535. For a simple normalization of values to within displayable intensity bounds, use KCOLOR_NORM_MAXCOLORS Alternatively, images can have contrast increased or reduced using on of the standard deviation normalizations. KCOLOR_NORM_1STDDEV will normalize values to within 1 standard deviation, KCOLOR_NORM_2STDDEV will normalize values to within 2 standard deviations, and KCOLOR_NORM_3STDDEV will normalize values to within 3 standard deviations.
XVW_COLOR_NORM_TYPE	Before colors are displayed by the color class, normalization must be done in order to ensure that the pixel values of the image fall within a visible range. Normalization can be one

Summary of Color Class Attributes	
Attribute	**Description**
XVW_COLOR_NORM_UBYTE	If desired, you may normalize unsigned byte data. Normally (especially when comparing images together) you don't want to perform normalization.
XVW_COLOR_NUMCOLORS	This *read only* attribute is used to obtain the size of the XColor array returned by XVW_COLOR_COLORS.
XVW_COLOR_PRIVATE_CMAP	To summarize very briefly, the color class will be able to allocate more colors if a private colormap is created. This allows for a better representation. However, since colormaps will not be shared with other applications, this may result in the annoying "techno-flashing" phenomenon where the colormaps of spectrum and any other applications that happen to be running at the same time are installed and de-installed as the pointer is moved in and out of the application's GUI. Thus, if this application wants as many colors to be allocated as possible, use of the private colormap allocation scheme is recommended. Technoflashing will increasingly become a problem as the application has to compete with other applications for color. Technoflashing is minimized by leaving an application with the private colormap set to FALSE, since the default colormap will be used. However, colors displayed may not truly reflect their real values. The recommended solution is to use a logical command line argument for applications, which will allow the user to change the value of the XVW_COLOR_PRIVATE_CMAP attribute or set thru the app-defaults.

Summary of Color Class Attributes	
Attribute	**Description**
XVW_COLOR_RED_FUNCTION	With images having colormaps made up of more than three columns, such as those produced by clustering algorithms, it is often informative to be able to apply a function to the values in those map columns in order to define the values that will be displayed as each of the red, green, and blue columns. For example, in an image having 6 map columns, you might define the red intensity of pixel 10 as (map column 2)[10]+ (map column 3) [10]/ (map column 4) [10]. In this case, the XVW_COLOR_RED_FUNCTION attribute may be set to the string defining the function to be applied in order to produce the values that will be used as red. Functions must have *only* variable *M*, where *M* stands for "map column"; following each *M* must be a number *starting at zero* and ranging to *N-1* where *N* is the number of map columns available in the image. For example, a valid function for red might be: (M2 - M3)/(M2 + M3) or (M0 + M1 + M2)/(M3 - M4) Equation evaluation follows the standard rules of precedence; evaluation proceeds from left to right, and use of parentheses is fully supported. In a more simple use, the XVW_COLOR_RED_FUNCTION may also be the variable representing the map column which is to be used to specify the red values of the pixels in the image. For example, if you wanted the fourth map column to define the red values of the pixels in the image, (remember that map column numbering begins at 0), you could set XVW_COLOR_RED_FUNCTION to: M3 You can also set the XVW_COLOR_RED_FUNCTION to a constant, if desired. For example, if you wanted the red values of all the pixels in the image to be 200, you could set XVW_COLOR_RED_FUNCTION to: 200 By default, the XVW_COLOR_RED_FUNCTION attribute is set to *M0*, or the first map column; thus, it produces the results that you would normally expect for an image with a colormap where no function was being applied.

Summary of Color Class Attributes	
Attribute	**Description**
XVW_COLOR_RED_MAPCOL	An image with a simple colormap has three map columns associated with it, where the pixels in the image are used to index into the map columns; the first map column defines the red values, the second map column defines the green values, and the third map column defines the blue values. In this way, the color for each pixel in the image is defined. Some images, however, have more than three map columns; for example, output images produced by clustering algorithms may have many map columns, none of which is necessarily associated only with red, green, or blue values. When multiple map columns are present, it may be useful to view any of the map columns as red, green, or blue. Thus, the XVW_COLOR_RED_MAPCOL attribute specifies the index of the map column that is to be used to specify the red values for each of the pixels in the image. Valid values are from *0* to *N-1*, where *N* is the number of map columns contained with the image. By default, XVW_COLOR_RED_MAPCOL is set to 0, indicating the first map column; thus, it produces the results that you would normally expect for an RGB image where only three map columns were present. Note that setting XVW_COLOR_RED_MAPCOL to the integer representing the desired map column is the same as setting the XVW_COLOR_RED_FUNCTION attribute to the variable *Mx*, where *x* is the index of the desired map column. For example, setting XVW_COLOR_RED_MAPCOL to 5 is the same as setting XVW_COLOR_RED_FUNCTION to "M5".

Summary of Color Class Attributes	
Attribute	**Description**
XVW_COLOR_RELOAD	Use of this *action attribute* causes the data to be re-read from the associated data object; the data is then redisplayed. Use of this attribute is useful when displaying data with a large number of image values. When displaying such data on screens that are less than 24 bit, the limitation of the number of colorcells available for allocation on the screen can cause difficulties in realistically displaying the colors in the data. If the data being displayed has more values than can be accomodated by the screen, then some data values must be mapped to the same pixel as other data values. Consider the situation of a large image that cannot be displayed in its entirety, and furthermore containing a much greater number of image values than can be displayed on the current screen, *not all of which appear in every region of the image*. The displayed result will be most acceptable for the region of the image was loaded first, since the image values in that region of the image will have the greatest chance of getting unique pixel values allocated for them. However, other regions of the image with a large number of data values that differ from the values of the first region may suffer from the effect of having a large number of data values all being mapped to the same pixel value. Such situations can be rectified by allowing the currently region to be re-read from scratch, causing the color allocation process to be redone, and the current region being allowed to commandeer all the available pixel values.
XVW_COLOR_SAVEMAP	This *action attribute* allows you to save the colormap associated with the displayed data object *by itself*; that is, the data object written out will not contain value, mask, time, or location data, but only the map data.

Descriptions of Color Class Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_COLOR_ALLOC_POLICY (N/A)	int	KCOLOR_ALLOC_READONLY	KCOLOR_ALLOC_READONLY KCOLOR_ALLOC_READWRITE
XVW_COLOR_BEGIN (N/A)	int	0	0 to height of colormap
XVW_COLOR_BLUE_FUNCTION (N/A)	char *	"M2"	see description

Descriptions of Color Class Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_COLOR_BLUE_MAPCOL (N/A)	int	0	valid map column index
XVW_COLOR_CALLBACK (N/A)	kfunc_void	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_COLOR_CHANGE_BLUE_MAPCOL (N/A)	int	N/A (action attribute)	TRUE
XVW_COLOR_CHANGE_GREEN_MAPCOL (N/A)	int	N/A (action attribute)	TRUE
XVW_COLOR_CHANGE_RED_MAPCOL (N/A)	int	N/A (action attribute)	TRUE
XVW_COLOR_COLORFILE (N/A)	char *	NULL	name of file w/ colormap
XVW_COLOR_COLOROBJ (N/A)	kobject	NULL	valid data object
XVW_COLOR_COLORS (N/A)	XColor *	NULL	
XVW_COLOR_END (N/A)	int	0	0 to height of colormap
XVW_COLOR_GREEN_FUNCTION (N/A)	char *	"M1"	see description
XVW_COLOR_GREEN_MAPCOL (N/A)	int	0	valid map column index
XVW_COLOR_NORM_METHOD (colorNormMethod)	int	KCOLOR_NORM_MAXCOLORS	KCOLOR_NORM_MAXCOLORS KCOLOR_NORM_1STDDEV KCOLOR_NORM_2STDDEV KCOLOR_NORM_3STDDEV KCOLOR_NORM_BESTGUESS KCOLOR_NORM_NONE
XVW_COLOR_NORM_TYPE (colorNormType)	int	KCOLOR_NORM_LOCAL	KCOLOR_NORM_GLOBAL KCOLOR_NORM_LOCAL
XVW_COLOR_NORM_UBYTE (colorNormUbyte)	int	TRUE	TRUE/FALSE
XVW_COLOR_NUMCOLORS (N/A)	int	0	value > 0
XVW_COLOR_PRIVATE_CMAP (colorPrivateCmap)	int	FALSE	TRUE/FALSE
XVW_COLOR_RED_FUNCTION (N/A)	char *	"M0"	see description

Descriptions of Color Class Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_COLOR_RED_MAPCOL (N/A)	int	0	valid map column index
XVW_COLOR_RELOAD (N/A)	int	N/A	TRUE (action attribute)
XVW_COLOR_SAVEMAP (N/A)	char *	NULL	valid filename

C. Visual Objects Related to Imaging

C.1. The Animate Object

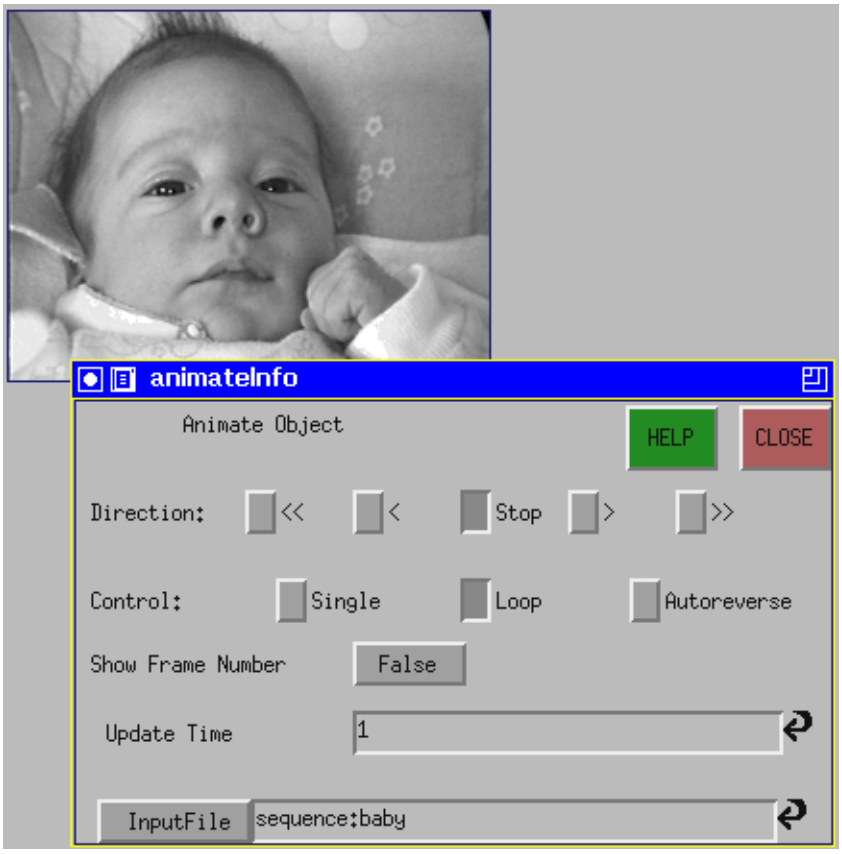


Figure 1: Here, the "sequence:baby" animation is displayed in an animation object, with the internal menuform of the animation object displayed. The animation menuform supports modification of the various animation attributes, such as sequence direction, frame number display, update time, and so on.

C.1.1. xvw_create_animate() — *create a slide animation visual object*

Synopsis

```
xvobject xvw_create_animate(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the animate object (for use in app-defaults files, etc)

Returns

The animate object on success, NULL on failure

Description

An animation visual object provides a mechanism with which a sequence of images may be animated. When stored in a K2 KDF file, image frames may be organized down depth, time, or elements. Attributes offer control over the speed with which the image frames are changed, and the direction of the animation. Animations may run in a single loop (one pass through the frames), a continuous loop (continual forward passes through the frames), or an autoreverse loop (continual passes forward through the frames, first forwards, then backwards).

C.1.2. Attributes of the Animation Object

Summary of Animate Attributes

Attribute	Description
`XVW_ANIMATE_CALLBACK`	If desired, a callback may be installed on the animate object that will be fired each time a new image in the sequence is displayed. The frame number will be passed in as the call_data; cast this parameter to an integer before using, as in: `int frame_number = (int) call_data;`

Summary of Animate Attributes	
Attribute	**Description**
XVW_ANIMATE_CONTROL	This attribute allows you to control how the sequencing is or KANIMATE_DIRECTION_REVERSE. The animation may be performed once completely, repeated indefinitely, or performed in one direction then reversed and repeated in the other direction. KANIMATE_CONTROL_LOOP ("Loop" on the "Control" selection of the menuform) does a full sequence through the animation, in the current animation direction. As soon as the sequence is finished, it is started again, so that the animation is put into a loop. This procedure will repeat until the animation is stopped. KANIMATE_CONTROL_SINGLE ("Single" on the "Control" selection of the menuform) does a single complete sequence of the animation, in the current animation direction, and then stops. ANIMATE_CONTROL_AUTOREVERSE ("Reverse" on the "Control" selection of the menuform) does a single complete sequence of the animation, in the current animation direction, then reverses the direction and sequences back. This procedure will repeat until the animation is stopped.
XVW_ANIMATE_DIRECTION	This attribute provides control over the direction in which the images are sequenced. Values include: KANIMATE_DIRECTION_STOP ("Stop" on the "Direction" toggle of the menuform) causes the animation to stop. KANIMATE_DIRECTION_PREVIOUS, ("<" on the "Direction" toggle of the menuform) advances a single frame in a backwards direction (frame N to N-1). KANIMATE_DIRECTION_NEXT, (">" on the "Direction" toggle of the menuform) advances a single frame in a forward direction (from frame N to N+1). KANIMATE_DIRECTION_REVERSE, ("<<" on the "Direction" toggle of the menuform) causes the animation to sequence in a backward direction. KANIMATE_DIRECTION_FORWARD, (">>" on the "Direction" toggle of the menuform) causes the animation to sequence in a forward direction.
XVW_ANIMATE_UPDATETIME	The time interval, in seconds, which will elapse before the animation object is updated with the next image in the sequence.

Descriptions of Animate Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_ANIMATE_CALLBACK (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_ANIMATE_CONTROL (animateControl)	int	KANIMATE_CONTROL_LOOP	KANIMATE_CONTROL_LOOP KANIMATE_CONTROL_SINGLE KANIMATE_CONTROL_AUTOREVERSE
XVW_ANIMATE_DIRECTION (animateDirection)	int	KANIMATE_DIRECTION_STOP	KANIMATE_DIRECTION_STOP KANIMATE_DIRECTION_PREVIOUS KANIMATE_DIRECTION_NEXT KANIMATE_DIRECTION_REVERSE KANIMATE_DIRECTION_FORWARD
XVW_ANIMATE_UPDATETIME (animateUpdatetime)	double	1.0	value > 0.0

C.1.3. Resource Set of the Animation Object

The inheritance tree of the animation object is as follows:

manager -> graphics -> color -> image -> animate

Accordingly, the complete resource set for the animation object includes:

1. The animate object attribute resource set, given above
2. The image object attribute resource set, given in Section C.2, "Attributes of the Image Object"
3. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
4. The general object attributes, given in Chapter 2, "The xvwidgets Library", Section B, "General Attributes of GUI and Visual Objects".

C.1.4. Example Using the Animate Visual Object

Examples using the animate visual object can be found in `$ENVISION/examples/animate/`. The simplest of these is as follows:

```
#include <envision.h>

/*
 *  This program does an animation sequence. The animation data
```

```
 *  is read in, the animation visual object is created, the visual object
 *  is associated with the animation data, and the animation is run.
 */
void main(
   int  argc,
   char *argv[])
{
     kobject  data_object;
     xvobject animate;
     char *filename = "sequence:baby";

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

        /* allow different images to be used, as in "% example sequence:bush" */
        if (argc > 1)
           filename = argv[1];

     /* create data object from the input file */
     data_object = kpds_open_input_object(filename);

     /* give it a colormap to make it look pretty */
     kcolor_set_attribute(data_object, KCOLOR_MAP_AUTOCOLOR, KRGB_SPIRAL);

        /* initialize the xvwidgets lib */
        if (!xvw_initialize(XVW_MENUS_XVFORMS))
        {
           kerror(NULL, "main", "unable to open display");
           kexit(KEXIT_FAILURE);
        }

     /*
      * create the animate object.  Set the animate object to operate on
         * the data object representing the input file, set the update time,
         * the animation control and the animation direction.   Note that
         * the XVW_ANIMATE_DIRECTION attribute is an "action attribute"; setting
         * it to KANIMATE_DIRECTION_FORWARD has the effect of starting the
         * animation in a forward direction.
      */
     animate = xvw_create_animate(NULL, "animate");
     xvw_set_attributes(animate,
                        XVW_ANIMATE_DIRECTION,  KANIMATE_DIRECTION_FORWARD,
                        XVW_IMAGE_IMAGEOBJ,     data_object,
                        XVW_ANIMATE_UPDATETIME, 0.2,
                        XVW_ANIMATE_CONTROL,    KANIMATE_CONTROL_AUTOREVERSE,
                        NULL);

     /* display & run */
     xvf_run_form();
}
```

C.2. The Image Object

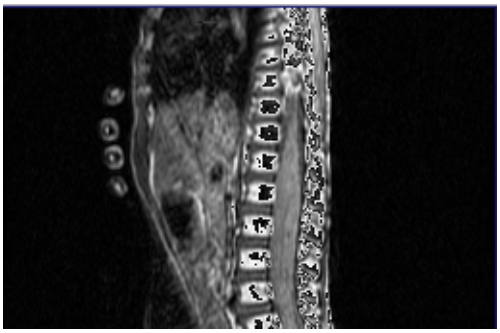


Figure 2: An image object is used to display a xray showing a cross-section of the human body. The internal menuforms of the image object (not shown here) allow the user to control the x and y position of the image window (for use with large images) as well as the image that is displayed.

C.2.1. xvw_create_image() — *create an image object*

Synopsis

```
xvobject xvw_create_image(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the image object (for use in app-defaults files, etc)

Returns

The image object on success, NULL on failure

Description

The image visual object provides a mechanism with which to easily display an image. If the input file has an associated colormap, that colormap will be used to display the image.

8 bit and 24 bit displays are fully supported by the image object; monochrome and 4 bit displays may be used but are not recommended.

Bit images, greyscale images, and RGB images are also supported. The image object is completely flexible as to the data type of the image data; all data types, including signed and unsigned bit, byte, short, int, float, double, and complex data types are supported.

The image object will react to changes in the input image; that is, if the contents of the input image change, the image object will immediately update to display the new image.

The image object supports the reading/writing of a variety of file formats, including:

Viff (VisiQuest 1.0.5 format)
Kdf (VisiQuest 2.1 data format)
Gdbm (VisiQuest 2.0 gdbm format)
Portable Any Map (PNM) format (including PPM, PGM, etc)
ASCII format
X Bitmap (XBM) format
X Pixel Map (XPM) format
X Window Dump (XWD) format
Application Visualization System (AVS) format

In addition to those listed above, the following formats are supported for write only:

Encapsulated PostScript (eps) format

In addition to those listed above, the following formats are supported for read only:

Raw (headerless) format

In the future, we also hope to support a number of additional formats, including:

Tagged Image File Format (TIFF)

C.2.2. Attributes of the Image Object

Summary of Image Attributes

Attribute	Description
`XVW_IMAGE_BACKING`	This attribute allows you set image backing store on (TRUE) or off (FALSE). Turning image backing store on causes that the image to be used as the background pixmap for the image display window. Image backing store allows you to drag annotations across an image, while reducing the annoying flicker that occurs as the refresh mechanism refreshes the parto of the image over which the annotation is being dragged. Be aware, however, that turning image backing store on implies an expensive operation every time the image itself is updated. Thus, you would not want to turn on backing store for an animation, or in any other circumstance when the image in question is going to be frequently updated.

Summary of Image Attributes	
Attribute	**Description**
XVW_IMAGE_BANDNUM	This is the band number of the data object which contains the image to be displayed. By default, band 0 (the first image) is displayed.

XVW_IMAGE_BAND_DIMENSIONS	This mask attribute dictates how the total number of displayable frames in the data object should be computed. The values are *or*'d in as desired. For example, if bands are sequenced with respect to depth and time:
	
	`int mask;`
	`mask = KIMAGE_DEPTH | KIMAGE_TIME;`
	`xvw_set_attribute(imageobj,`
	`XVW_IMAGE_BAND_DIMENSIONS, mask);`
	Note that use of this attribute implies foreknowledge about the dimensionality of the data to be displayed.

XVW_IMAGE_BAND_MAXNUM	The maximum index of the bands available in the currently displayed image; ie, the number of bands computed for the data, given the value of XVW_IMAGE_BAND_DIMENSIONS.
XVW_IMAGE_CLIPFILE	The name of a file containing a clip mask may be specified using this attribute; the default is NULL. Note that this attribute is mutually exclusive with XVW_IMAGE_CLIPOBJ; use one or the other, not both. See the explanation of XVW_IMAGE_CLIPOBJ for more details on the clip mask.
XVW_IMAGE_CLIPOBJ	A clip mask is used to obscure portions of the displayed image; if desired, this data object (kobject) attribute can be used to set a clip mask for the image. When an image is used as a clip mask for the displayed image, the only part of the image that will appear normally is that part defined by the pixels in the clip mask image that have a value of (1). All other parts of the current image will appear in the background color of the image window. Note that ONLY images of data type bit can be used as clip masks; if the input image to be used as a clip mask is not of type bit, it will be internally converted to type bit, without prompting or warning, before it is used. Note that this conversion will not affect the original file in any way. If a clip mask currently being used is no longer desired, set this attribute back to NULL. Note that this attribute is mutually exclusive with XVW_IMAGE_CLIPFILE; use one or the other, not both.
XVW_IMAGE_COLORMAPFILE	The name of a file containing the desired colormap may be specified using this attribute; the default is NULL. Note that this attribute is mutually exclusive with XVW_IMAGE_COLORMAPOBJ; use one or the other, not both. See the explanation of XVW_IMAGE_COLORMAPOBJ for more details on the color map.

Summary of Image Attributes	
Attribute	**Description**
XVW_IMAGE_COLORMAPOBJ	This attribute allows the assignment of a different color map to the image being displayed, other than the colormap stored with the image. The attribute expects a data object (kobject) containing a color map, and replaces the displayed image's color map with the new color map. Note that this attribute is mutually exclusive with XVW_IMAGE_COLORMAPFILE; use one or the other, not both.
XVW_IMAGE_COMPLEX_CONVERT	Only relevant if complex images are to be displayed, this attribute maps to the polymorphic data services KPDS_VALUE_COMPLEX_CONVERT attribute. It specifies how complex data should be converted in preparation for display as an image. See Chapter 3 of Programming Services Manual Volume I, "Math Services" for explanations of the various settings.
XVW_IMAGE_IMAGEFILE	Specifies the name of the file containing the image to be displayed. Note that this attribute is mutually exclusive with XVW_IMAGE_IMAGEOBJ; use one or the other, not both.
XVW_IMAGE_IMAGEOBJ	This is the data object (kobject) containing the image to be displayed. Note that this attribute is mutually exclusive with XVW_IMAGE_IMAGEFILE; use one or the other, not both.
XVW_IMAGE_OVERLAYFILE	The name of a file containing an overlay image may be specified using this attribute; the default is NULL. Note that this attribute is mutually exclusive with XVW_IMAGE_OVERLAYOBJ; use one or the other, not both. See the explanation of XVW_IMAGE_OVERLAYOBJ for more details on the overlay image.
XVW_IMAGE_OVERLAYOBJ	Under construction...
XVW_IMAGE_PIXEL	Corresponding to the current image value, as indicated by XVW_IMAGE_VALUE, this is the pixel value that was actually allocated for that particular image data value.
XVW_IMAGE_REDISPLAY	This state variable attribute causes the currently displayed image to be redisplayed. Use of this attribute may be appropriate after making a deliberate change to the image data of the data object being displayed.

Summary of Image Attributes	
Attribute	**Description**
XVW_IMAGE_RELOAD	When set to TRUE, this attribute causes the image data to be re-read from the associated data object; the data is then redisplayed. Use of this attribute is useful when displaying images with large images (large enough to require the use of a pan icon) that also have a large number of image values. When displaying such images on screens that are less than 24 bit, the limitation of the number of colorcells available for allocation on the screen can cause difficulties in realistically displaying the colors in the image. If the image being displayed has more values than can be accomodated by the screen, then some image values must be mapped to appear as the same pixel as other image values. If the image is small, or the number of image values is not *much* greater than that which can be displayed on the screen being used, there is not much that can be done. However, consider the situation of a large image that cannot be displayed in its entirety, and furthermore containing a much greater number of image values than can be displayed on the current screen, *not all of which appear in every region of the image*. The displayed result will be most acceptable for the region of the image was loaded first, since the image values in that region of the image will have the greatest chance of getting unique pixel values allocated for them. However, other regions of the image with a large number of data values that differ from the values of the first region may suffer from the effect of having a large number of data values all being mapped to the same pixel value. Such situations can be rectified by allowing the currently region to be re-read from scratch, causing the color allocation process to be redone, and the current region being allowed to commandeer all the available pixel values.
XVW_IMAGE_ROI	If this action attribute is *set*, then the region of interest contained in the data object specified will be inserted into the displayed image. Note that the polymorphic data services attribute KPDS_SUBOBJECT_POSITION may be set on the data object to specify where in the image the region of interest is to be inserted. If the KPDS_SUBOBJECT_POSITION attribute is *not* set in the data object, then the user will be interactively prompted to indicate where the region of interest is to be placed. If a *get* attribute procedure is done, then the user will be interactively prompted to specify the desired region of interest, and the region of interest will be stored in the specified data object. The KPDS_SUBOBJECT_POSITION will be set on the resulting data object, to indicate the position of origination in the displayed image.

Summary of Image Attributes	
Attribute	**Description**
XVW_IMAGE_ROI_MULTIBAND	When the XVW_IMAGE_ROI attribute is used to extract a region of interest, this attribute specifies whether the shape of the ROI should extract the present band or all bands within the ROI extent. of multiband being TRUE, all bands within the ROI will be extracted, otherwise only the current displayed band is extracted.
XVW_IMAGE_ROI_MULTIPLE	When the XVW_IMAGE_ROI attribute is used to extract a region of interest, this attribute specifies whether multiple ROIs should be extracted. ROI's may be rectangular, polygonal, or circular. In the case of multiple ROIs being TRUE, a single data object will be passed back with the mask set indicating which ROI belongs to which. So for the first roi, the mask will be set to 1. For the second roi, the mask will be set to set to 2, and so forth. If multiple ROIs is set to FALSE then a binary mask will be created, using 0's and 1's to represent which bits in the mask are valid for the single roi.
XVW_IMAGE_ROI_POLICY	When the XVW_IMAGE_ROI attribute is used to extract a region of interest, this attribute specifies whether the ROI is defined by the region inside the shape, by the region outside the shape, or by the outline of the shape itself.
XVW_IMAGE_ROI_PRESENTATION	When the XVW_IMAGE_ROI attribute is used to extract a region of interest, this attribute specifies whether the ROI should be handed back as a signal, image, or surface. If the roi is defined as a signal, then it can be displayed using the 2D plotting routines. If the roi is extracted as an image, then the roi can be displayed using the image display routines. And finally, if extracted as a surface, the roi can be displayed using the 3D plotting routines.
XVW_IMAGE_ROI_SHAPE	When the XVW_IMAGE_ROI attribute is used to extract or insert a region of interest, this attribute specifies the shape of the ROI. ROI's may be rectangular, polygonal, or circular.
XVW_IMAGE_SAVEIMAGE	This action attribute (use only with *xvw_set_attribute(s)*) allows you to write out the currently displayed image. Simply provide a filename for the image to be saved.
XVW_IMAGE_UPDATE_CALLBACK	If desired, a callback may be installed on the image object that will be fired each time a new filename is input to the image object for display. The filename will be passed in as the call_data; cast this parameter to an integer before using, as in: `kchar *filename = (char *) call_data;`
XVW_IMAGE_VALUE	This attribute is the image data value (cast to double) at the current location of the pointer in the image. Note that the current location of the pointer in the image can be obtained using XVW_IMAGE_XPOSITION and XVW_IMAGE_YPOSITION.
XVW_IMAGE_XMAGNIFY	The image x magnification factor specifies the amount in which to magnify the image in the horizontal direction. The magnification is double which if greater than 1.0 will enlarge the image; however, if less than 1.0 will shrink the image.

Summary of Image Attributes	
Attribute	**Description**
XVW_IMAGE_XOFFSET	The image x offset specifies the horizontal offset within the image of the image display window. Together, the x offset and y offset specify the upper left corner portion of the image that appears in the display window. For small images which fit entirely within the image display window, this will always be (0,0); however, for large images that must use a pan icon because the image will not fit entirely within the image window, the x offset may be any value from 0 to (width of image - image display window width).
XVW_IMAGE_XPOSITION	The x position reflects the current horizontal pointer position within the image. The x position is corrected to take into account the image offset (if any). Thus, regardless of whether or not a large image with a pan icon is being displayed, and a pan icon has been used to change the portion of the image that appears in the image window, the position returned will be the position of the pointer within the image *as a whole* (world coordinates), as opposed to the position of the pointer within the actual window in which the image is being displayed (device coordinates). Getting the value of this attribute is how to obtain the current location of the pointer in the image; setting the value of this attribute will cause any objects subordinate to the image (such as zoom, position, etc) to update their values to the new position.
XVW_IMAGE_YMAGNIFY	The image x magnification factor specifies the amount in which to magnify the image in the vertical direction. The magnification is double which if greater than 1.0 will enlarge the image; however, if less than 1.0 will shrink the image.
XVW_IMAGE_YOFFSET	The image y offset specifies the vertical offset within the image of the image display window. Together, the x offset and y offset specify the upper left corner portion of the image that appears in the display window. For small images which fit entirely within the image display window, this will always be (0,0); however, for large images that must use a pan icon because the image will not fit entirely within the image window, the y offset may be any value from 0 to (height of image - image display window height).
XVW_IMAGE_YPOSITION	The y position reflects the current vertical pointer position within the image. See the explanation of XVW_IMAGE_XPOSITION for more details.

Descriptions of Image Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_IMAGE_BACKING (imageBacking)	int	TRUE	TRUE/FALSE

Descriptions of Image Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_IMAGE_BANDNUM (N/A)	int	0	0 < bandnum < # bands in image
XVW_IMAGE_BAND_DIMENSIONS (N/A)	int	KIMAGE_ELEMENTS | KIMAGE_DEPTH | KIMAGE_TIME	Any combination of: KIMAGE_ELEMENTS | KIMAGE_DEPTH | KIMAGE_TIME
XVW_IMAGE_BAND_MAXNUM (N/A)	int	0	0 to (number of bands computed - 1).
XVW_IMAGE_CLIPFILE (N/A)	char *	NULL	valid name of input file containing clip mask
XVW_IMAGE_CLIPOBJ (N/A)	kobject	NULL	valid data object defining clip maks
XVW_IMAGE_COLORMAPFILE (N/A)	char *	NULL	valid name of input file containing colormap
XVW_IMAGE_COLORMAPOBJ (N/A)	kobject	NULL	valid data object defining colormap
XVW_IMAGE_COMPLEX_CONVERT (imageComplexConvert)	int	KLOGMAGP1	AccuSoftEAL KIMAGINARY KPHASE KMAGNITUDE KLOGMAGP1 KLOGMAG KLOGMAGSQP1 KLOGMAGSQ KMAGSQ
XVW_IMAGE_IMAGEFILE (N/A)	char *	NULL	valid name of input file containing image to display
XVW_IMAGE_IMAGEOBJ (N/A)	kobject	NULL	valid data object defining image to display
XVW_IMAGE_OVERLAYFILE (N/A)	char *	NULL	valid name of input file containing overlay image
XVW_IMAGE_OVERLAYOBJ (N/A)	kobject	NULL	valid data object defining an overlay image
XVW_IMAGE_PIXEL (N/A)	unsigned long	0	pixel value
XVW_IMAGE_REDISPLAY (N/A)	int	FALSE	TRUE/FALSE
XVW_IMAGE_RELOAD (N/A)	int	FALSE	TRUE/FALSE
XVW_IMAGE_ROI (N/A)	kobject	NULL	valid data object

Descriptions of Image Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_IMAGE_ROI_MULTIBAND (imageRoiMultiband)	int	TRUE	TRUE/FALSE
XVW_IMAGE_ROI_MULTIPLE (imageRoiMultiple)	int	FALSE	TRUE/FALSE
XVW_IMAGE_ROI_POLICY (imageRoiPolicy)	int	KIMAGE_ROI_INSIDE	KIMAGE_ROI_INSIDE KIMAGE_ROI_OUTLINE KIMAGE_ROI_OUTSIDE
XVW_IMAGE_ROI_PRESENTATION (imageRoiPresentation)	int	KIMAGE_ROI_IMAGE	KIMAGE_ROI_SIGNAL KIMAGE_ROI_IMAGE KIMAGE_ROI_SURFACE
XVW_IMAGE_ROI_SHAPE (imageRoiShape)	int	KIMAGE_ROI_RECTANGLE	KIMAGE_ROI_RECTANGLE KIMAGE_ROI_POLYLINE KIMAGE_ROI_CIRCLE KIMAGE_ROI_ELLIPSE KIMAGE_ROI_LINE KIMAGE_ROI_FREEHAND
XVW_IMAGE_SAVEIMAGE (N/A)	char *	NULL	filename for saved image
XVW_IMAGE_UPDATE_CALLBACK (N/A)	void (*call-back_rou-tine)(xvob-ject, kaddr, kaddr)	NULL	void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_IMAGE_VALUE (N/A)	double	0.0	pixel value
XVW_IMAGE_XMAGNIFY (imageXmagnify)	double	1.0	any value greater than 0.0
XVW_IMAGE_XOFFSET (imageXoffset)	int	-1	-1 <= value <= image width
XVW_IMAGE_XPOSITION (N/A)	int	N/A	0 =< value <= image width
XVW_IMAGE_YMAGNIFY (imageYmagnify)	double	1.0	any value greater than 0.0
XVW_IMAGE_YOFFSET (imageYoffset)	int	-1	-1 <= value <= image height
XVW_IMAGE_YPOSITION (N/A)	int	N/A	0 =< value <= image height

C.2.3. Complete Resource Set of the Image Object

The inheritance tree of the image object is as follows:

manager -> graphics -> color -> image

Accordingly, the complete resource set for the image object includes:

1. The image attribute resource set, given above

2. The color attribute resource set, given in Section B, "The Color Attributes"

3. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"

4. The general object attributes, given in Chapter 2, "The xvwidgets Library", Section B, "General Attributes of GUI and Visual Objects".

C.2.4. Example using the Image Object

A number of example programs using the image visual object can be found in `$ENVISION/examples/image`. The simplest of these is as follows.

```
#include <envision.h>

/*
 *  This simple introductory program creates a window with an image object; the
 *  image displayed is the mandril image, which is specified using the keyword
 *  syntax rather than the absolute path to the image.
 */
void main(
   int  argc,
   char *argv[])
{
     xvobject parent, image, position;
/*   char *filename = "image:mandril-rgb";*/
     char *filename = "image:ball";

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

     /* Initialize the xvwidgets library */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror("test", "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /*
      *  Create the image display.  Since the parent is NULL, a toplevel
      *  window will be created automatically, and the image object placed
         *  inside.  The xvw_set_attribute() call is used to specify the image
         *  file to be displayed.
      */
     parent = xvw_create_manager(NULL, "parent");

     image = xvw_create_image(parent, "image");
     xvw_set_attribute(image, XVW_IMAGE_IMAGEFILE, filename);

     position = xvw_create_position(parent, "position");
```

```
        xvw_set_attributes(position,
                    XVW_POSITION_FILENAME, filename,
                    XVW_BELOW, image,
                    XVW_LEFT_OF, NULL,
                    XVW_RIGHT_OF, NULL,
                    NULL);

        /* display & run; there is no way to exit the program but ^C */
        xvf_run_form();
    }
```

C.3. The ImageIcon Object

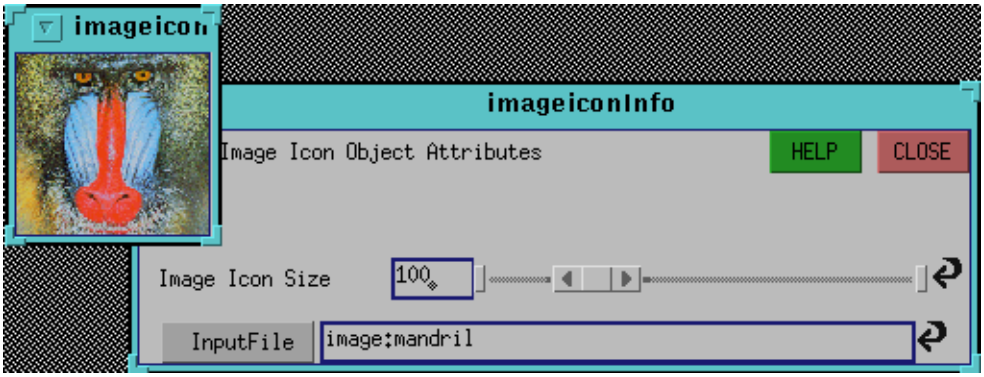


Figure 3: An image icon object is used to display color mandril as an icon. The internal menuforms of the image icon object allow the user to control the icon size as well as the image that is displayed.

C.3.1. xvw_create_imageicon() — *create a imageicon object*

Synopsis

```
xvobject xvw_create_imageicon(
    xvobject parent,
    char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the imageicon object (for use in app-defaults files, etc)

Returns

Returns the created imageicon xvobject, or NULL upon failure

Description

The image icon visual object displays the image as an icon (a miniature, subsampled version of the image).

C.3.2. Attributes of the ImageIcon Object

Summary of ImageIcon Attributes	
Attribute	**Description**
`XVW_IMAGEICON_SIZE`	This integer value specifies the size of the icon in pixels. The image icon size should always be used rather than setting width and height separately; otherwise, the image may not be proportionally correct.

Descriptions of ImageIcon Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
`XVW_IMAGEICON_SIZE` (`imageiconSize`)	`int`	`100`	value > 0

C.3.3. Complete Resource Set of the ImageIcon Object

The inheritance tree of the imageicon object is as follows:

manager -> graphics -> color -> image -> imageicon

Accordingly, the complete resource set for the imageicon object includes:

1. The imageicon attribute resource set, given above
2. The image attribute resource set, given in section C.2, "Attributes of the Image Object"
3. The color attribute resource set, given in Section B, "The Color Attributes"
4. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
5. The general object attributes, given in Chapter 2, "The xvwidgets Library", Section B, "General Attributes of GUI and Visual Objects".

C.3.4. Example using the ImageIcon Visual Object

An example program using the imageicon visual object can be found in `$ENVISION/examples/imgicon/1.create_imgicon`. This program is as follows.

```
#include <envision.h>

/*
 * This simple program creates an imageicon object from the lizard image.
 */
void main(
   int  argc,
   char *argv[])
{
     xvobject imageicon;
     char *filename = "image:mandril";

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

     /* Initialize the xvwidgets lib. */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /* create the image icon with a size of (75 x 75) pixels */
     imageicon = xvw_create_imageicon(NULL, "imageicon");
     xvw_set_attributes(imageicon,
                     XVW_IMAGEICON_SIZE,  50,
                     XVW_IMAGE_IMAGEFILE, filename,
                     NULL);

     /* display & run; there is no way to exit the program but ^C */
     xvf_run_form();
}
```

C.4. The PanIcon Object

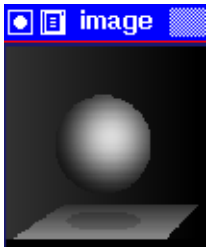


Figure 4: A pan icon allows the user to pan around an image that is bigger than the area in which it is displayed. This pan icon is panning on the "ball" image ($SAMPLEDATA/data/images/ball.xv, shorthand "image:ball").

C.4.1. xvw_create_panicon() — *create a panicon object*

Synopsis

```
xvobject xvw_create_panicon(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the panicon object (for use in app-defaults files, etc)

Returns

Returns the panicon object, or NULL upon failure

Description

A panicon object provides a mechanism with which to roam around in an image which is too large to be accomodated by the image object in which it is displayed. Thus, the panicon object must be used in conjunction with an image object on which it will pan.

Note that the `XVW_IMAGE_IMAGEOBJ` attribute is used to specify the data object on which the panicon should pan; this should be the same data object as the one displayed in the image object.

The panicon object will *not* be mapped unless it is needed; a created panicon object will remain "invisible" if the image being displayed is small enough to fit within the image object. Thus, if an application using an image object expects to *ever* receive a request to display an image which is larger than the image object size, a pan icon object should *always* be created; it will not appear unless it is necessary.

C.4.2. Attributes of the PanIcon Object

Summary of PanIcon Attributes	
Attribute	**Description**

Summary of PanIcon Attributes	
Attribute	**Description**
XVW_PANICON_CALLBACK	If desired, a callback may be installed on the panicon object that will be fired each time the user repositions the pan box in the pan icon. Note that no call_data is sent to the callback; to obtain the new position of the pan box with respect to the displayed image, use XVW_IMAGE_XOFFSET and XVW_IMAGE_YOFFSET on the associated image object.
XVW_PANICON_HEIGHT	The height in the displayed image defining the region on which the panicon is panning (this region is outlined by a white window in the panicon).
XVW_PANICON_SIZE	The desired physical size of the pan icon. Note that the actual size of the pan icon may be automatically modified to preserve proportionality in the event that a non-square image is displayed.
XVW_PANICON_VISIBILITY_CALLBACK	PanIcon objects are created for use in conjunction with an image object. When the image to be displayed is too large to fit in the window of the image object, the pan icon will be mapped; when the image to be displayed is small enough to fit entirely in the image window, the pan icon is unmapped. that will be fired when the pan icon is mapped or unmapped. A flag indicating whether the pan icon has just been mapped or unmapped will be passed in as the call_data; cast this parameter to an integer before using, as in: `int mapped = (int) call_data;` If the flag is TRUE, this implies that the pan icon has just been mapped and is now visible. If the flag is FALSE, this implies that the pan icon has just been unmapped and is no longer visible. This callback is for use primarily so that the calling application can adjust its GUI if desired, to accomodate the presence or absence of the pan icon.
XVW_PANICON_WIDTH	The width in the displayed image defining the region on which the panicon is panning (this region is outlined by a white window in the panicon).
XVW_PANICON_XPOS	The X position in the displayed image defining the region on which the panicon is panning (this region is outlined by a white window in the panicon).
XVW_PANICON_YPOS	The Y position in the displayed image defining the region on which the panicon is panning (this region is outlined by a white window in the panicon).

Descriptions of PanIcon Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**

Descriptions of PanIcon Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_PANICON_CALLBACK (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_PANICON_HEIGHT (N/A)	int	100	values > 50
XVW_PANICON_SIZE (paniconSize)	int	100	values > 50
XVW_PANICON_VISIBILITY_CALLBACK (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_PANICON_WIDTH (N/A)	int	100	values > 50
XVW_PANICON_XPOS (N/A)	int	0	0 - screen width
XVW_PANICON_YPOS (N/A)	int	0	0 - screen height

C.4.3. Resource Set of the PanIcon Object

The inheritance tree of the panicon object is as follows:

manager -> graphics -> color -> image -> panicon

Accordingly, the complete set of attributes for the panicon object includes:

1. The panicon object attribute resource set, given above
2. The image object attribute resource set, given in Section C.2 "Attributes of the Image Object"
3. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
4. The general object attributes, given in Chapter 2, "The xvwidgets Library", Section B, "General Attributes of GUI and Visual Objects".

C.4.4. Example Using the PanIcon Visual Object

Examples using the panicon visual object can be found in `$ENVISION/examples/panicon/`. The simplest of these is as follows:

```
#include <envision.h>

/*
 * This example displays a big image which needs a pan icon; the image and
 * the pan icon are created in separate windows. Since NULL is specified as
 * the parent for both the image and the pan icon, they are displayed in
 * independent windows.
 *
 * NOTE: This example will crash on an OSF 3.2 architecture;  there are
 *       points in the "clust:tmi" data set that result in a machine
 *       conversion problem in casting to double.
 */
static void image_update_cb PROTO((xvobject, kaddr, kaddr));
xvobject panicon;

void main(
   int  argc,
   char *argv[])
{
     kobject  data_object;
     xvobject image;
     char *filename = "image:ball";
     int  w, h, d, t, e;
     int  desired_width  = 810;
     int  desired_height = 760;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /*
      * If we input an image that is smaller than 2000 x 2000, scale it UP
         * to that size, so that we will actually need a pan icon.  The pan
      * icon will not display unless it is needed; it is considered needed
         * only when the image object is able to to display only a portion of
         * it.  In general, you *don't* want to do this!
      */
     data_object = kpds_open_object(filename, KOBJ_READ);
        kpds_get_attribute(data_object, KPDS_VALUE_SIZE, &w, &h, &d, &t, &e);
        w = kmax(w, desired_width);
        h = kmax(h, desired_height);
        kpds_set_attributes(data_object,
                    KPDS_VALUE_SIZE, w, h, d, t, e,
                    KPDS_VALUE_INTERPOLATE, KZERO_ORDER,
                    NULL);

     image = xvw_create_image(NULL, "image");
     xvw_set_attribute(image, XVW_IMAGE_IMAGEOBJ, data_object);
```

```
     /* add callback to update panicon when input file changes */
        xvw_add_callback(image, XVW_IMAGE_UPDATE_CALLBACK,
                         image_update_cb, NULL);


     /*
      * create the pan icon;  it will only be displayed when it is necessary;
      * although in this program we have forced it to be necessary.
         * note that setting the parent to NULL forces the pan icon to have
         * a window that is independent from the window in which the image is
         * displayed.  more often, it is useful to have a single backplane
         * hold both the image and the pan icon.
      */
     panicon = xvw_create_panicon(image, "panicon");
     xvw_set_attribute(panicon, XVW_IMAGE_IMAGEOBJ, data_object);

     /* display & run; there is no way to exit the program but ^C */
     xvf_run_form();
}

/*
 *  callback to update panicon object and position object to operate on the
 *  new image when the image object's internal menuform is used to input a
 *  new image.
 */
static void image_update_cb(
    xvobject object,
    kaddr    client_data,
    kaddr    call_data)
{
        char *filename = (char *) call_data;
        kobject data_object;

        /*
         * get data object from the modified file
         */
        if ((data_object = kpds_open_object(filename, KOBJ_READ)) != NULL)
        {
            xvw_set_attribute(panicon,  XVW_IMAGE_IMAGEOBJ,  data_object);
            kpds_close_object(data_object);
        }

}
```

C.5. The Position Object

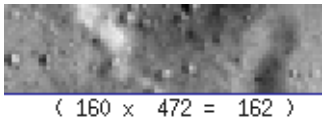


Figure 5: The position object tracks the movement of the pointer in an image object. Here, a position object is located beneath an image object.

C.5.1. xvw_create_position() — *create a position object*

Synopsis

```
xvobject xvw_create_position(
   xvobject parent,
   char   *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the position object (for use in app-defaults files, etc)

Returns

The position object on success, NULL on failure

Description

A position object is used to track movement of the pointer in an image; thus, it was designed to be used in conjunction with an image object.

The position object displays values in an (X x Y) or (X x Y = Z) format, where X and Y display the location of the pointer in the image object, and the optional Z is the pixel value of the image at that location. When the latter format is used with an RGB image, the format is modified to be (X x Y = R G B).

C.5.2. Attributes of the Position Visual Object

Summary of Position Attributes	
Attribute	**Description**
XVW_POSITION_FILENAME	A file containing the image on which the position object will track may be specified directly using this attribute.
XVW_POSITION_OBJECT	This is the data object (kobject) containing the image on which the position object will track. Note that this attribute is mutually exclusive with XVW_POSITION_FILENAME; use one or the other, not both.
XVW_POSITION_SHOW_VALUE	If TRUE, the position object will display information in (X x Y = Z) format, where X and Y display the location of the pointer in the data object, and Z is the pixel value at that location. If FALSE, the position object will simply display (X x Y) location information; the pixel value will be omitted.
XVW_POSITION_UPDATEMODE	The update mode attribute indicates when the position object is to update the information which it displays. If set to KPOSITION_UM_CONTINUOUS, the position object will continually update as the pointer is moved across the image; if set to KPOSITION_UM_BUTTON_PRESS, the position object will not update until the mouse button is pressed in the image object.

Descriptions of Position Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_POSITION_FILENAME (N/A)	char *	NULL	valid input filename
XVW_POSITION_OBJECT (N/A)	kobject	NULL	valid data object
XVW_POSITION_SHOW_VALUE (positionShowValue)	int	TRUE	TRUE/FALSE
XVW_POSITION_UPDATEMODE (positionUpdatemode)	int	KPOSITION_UM_CONTINUOUS	KPOSITION_UM_CONTINUOUS KPOSITION_UM_BUTTON_PRESS

C.5.3. Complete Resource Set of the Position Visual Object

The inheritance tree of the position object is as follows:

manager -> graphics -> string -> position

Accordingly, the complete resource set for the position object includes:

1. The position attribute resource set, given above

2. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"

3. The general object attributes, given in Chapter 2, "The xvwidgets Library", Section B, "General Attributes of GUI and Visual Objects".

C.6. The PrintPixel Object

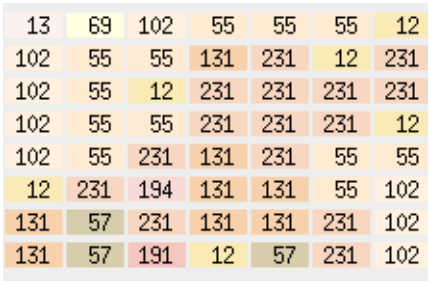


Figure 6: A printpixel object prints the pixel values in an image as the pointer is moved over the image. If desired, the background color over which each pixel value is printed may appear in the color of the pixel.

C.6.1. xvw_create_printpixel() — *create a printpixel xvobject*

Synopsis

```
xvobject xvw_create_printpixel(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the printpixel object (for use in app-defaults files, etc)

Returns

The printpixel xvobject on success, NULL otherwise

Description

A printpixel object provides a mechanism which will track pointer movement in an image object and

print the values of the pixels in the area under the pointer. It was designed for use in conjunction with an image object.

The printpixel object is made up of one or more (width x height) sets of label objects. The background of each label is the color in which the pixel appears.

C.6.2. Attributes of the PrintPixel Object

Summary of PrintPixel Attributes

Attribute	Description
XVW_PRINTPIXEL_CLIPFILE	The name of a file containing a clip mask may be specified using this attribute; the default is NULL. Note that this attribute is mutually exclusive with XVW_PRINTPIXEL_CLIPOBJ; use one or the other, not both. See the explanation of XVW_PRINTPIXEL_CLIPOBJ for more details on the clip mask.
XVW_PRINTPIXEL_CLIPOBJ	A clip mask is used to obscure portions of the displayed pixels; if desired, this data object (kobject) attribute can be used to set a clip mask for the data. When a clip mask is used with the printpixel object, the only pixel values that will be displayed by the printpixel object will be those that correspond to a pixel value of (1) in the clip mask. Pixels corresponding to a value of (0) in the clip mask will not be displayed. Note that ONLY images of data type bit can be used as clip masks; if the input image to be used as a clip mask is not of type bit, it will be internally converted to type bit, without prompting or warning, before it is used. Note that this conversion will not affect the original file in any way. If a clip mask currently being used is no longer desired, set this attribute back to NULL. Note that this attribute is mutually exclusive with XVW_PRINTPIXEL_CLIPFILE; use one or the other, not both.
XVW_PRINTPIXEL_DEPTH_OFFSET	When the input data object has depth > 1, this attribute may be used to specify the depth offset at which the pixel values are to be extracted (the printpixel object always operates on a (width x height) region).
XVW_PRINTPIXEL_ELEMENTS_OFFSET	When the input data object has elements > 1, this attribute may be used to specify the elements offset at which the pixel values are to be extracted (the printpixel object always operates on a (width x height) region).
XVW_PRINTPIXEL_FILENAME	The file containing the data object for which the pixel values are to be displayed. Note that this attribute is mutually exclusive with XVW_PRINT-PIXEL_OBJECT; use one or the other, not both.
XVW_PRINTPIXEL_HEIGHT	Specifies the number of pixel values that should be displayed in the vertical direction on the printpixel display.

Summary of PrintPixel Attributes	
Attribute	**Description**
XVW_PRINTPIXEL_OBJECT	This is the data object (kobject) containing the image for which pixel values are to be displayed. Note that this attribute is mutually exclusive with XVW_PRINTPIXEL_FILENAME; use one or the other, not both.
XVW_PRINTPIXEL_SHOWCOLOR	Indicates whether or not the color of the pixel under the pointer should be displayed as the background color for the label object in which the pixel values are displayed. When set to FALSE, the background will be black.
XVW_PRINTPIXEL_TIME_OFFSET	When the input data object has time > 1, this attribute may be used to specify the time offset at which the pixel values are to be extracted (the printpixel object always operates on a (width x height) region).
XVW_PRINTPIXEL_UPDATEMODE	Indicates whether the printmapval display is to be updated continuously as the pointer moves across the image, or not until the button is clicked at a particular location in the image.
XVW_PRINTPIXEL_WIDTH	Specifies the number of pixel values that should be displayed in the horizontal direction on the printpixel display.
XVW_PRINTPIXEL_XPOSITION	This is the X position in the image corresponding to the map value that appears in the upper left hand corner of the printmapval grid.
XVW_PRINTPIXEL_YPOSITION	This is the Y position in the image corresponding to the map value that appears in the upper left hand corner of the printmapval grid.

Descriptions of PrintPixel Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_PRINTPIXEL_CLIPFILE (N/A)	char *	NULL	valid name of input file containing clip mask
XVW_PRINTPIXEL_CLIPOBJ (N/A)	kobject	NULL	valid data object defining clip maks
XVW_PRINTPIXEL_DEPTH_OFFSET (N/A)	int	0	0 < value < depth of data object value segment
XVW_PRINTPIXEL_ELEMENTS_OFFSET (N/A)	int	0	value < number of elements in data object value segment
XVW_PRINTPIXEL_FILENAME (N/A)	char *	NULL	valid input filename
XVW_PRINTPIXEL_HEIGHT (printpixelHeight)	int	9	1 <= height <= image height
XVW_PRINTPIXEL_OBJECT (N/A)	kobject	NULL	valid data object
XVW_PRINTPIXEL_SHOWCOLOR (printpixelShowcolor)	int	FALSE	TRUE/FALSE
XVW_PRINTPIXEL_TIME_OFFSET (N/A)	int	0	value < time size of data object value segment

Descriptions of PrintPixel Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_PRINTPIXEL_UPDATEMODE (printpixelUpdatemode)	int	KPRINTPIXEL_UM_CONTINUOUS	KPRINTPIXEL_UM_CONTINUOUS KPRINTPIXEL_UM_BUTTONPRESS
XVW_PRINTPIXEL_WIDTH (printpixelWidth)	int	7	1 <= width <= image width
XVW_PRINTPIXEL_XPOSITION (N/A)	int	0	0 - image width
XVW_PRINTPIXEL_YPOSITION (N/A)	int	0	0 - image height

C.6.3. Resource Set of the PrintPixel Object

The inheritance tree of the printpixel object is as follows:

manager -> graphics -> color -> printpixel

Accordingly, the complete resource set for the printpixel object includes:

1. The printpixel object attribute resource set, given above
2. The color attribute resource set, given in Section B, "The Color Attributes"
3. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
4. The general object attributes, given in Chapter 2, "The xvwidgets Library", Section B, "General Attributes of GUI and Visual Objects".

C.6.4. Example Using the PrintPixel Visual Object

Examples using the printpixel visual object can be found in `$ENVISION/examples/color/printpixel/`. The simplest of these is as follows:

```
#include <envision.h>

/*
 *  This program creates an image object, a position object to reflect the
 *  location of the mouse pointer in the image, and a printpixel object to
 *  print the values of the pixels surrounding that location.
 *
 *  Note that you do NOT have to write an event handler to make the
 *  printpixel object update; the fact that we have created an image object
 *  using the same file for it's data does the trick.  The image object will
 *  recognise pointer motion within it, and cause the printpixel object (and
 *  the position object as well) to be updated automatically.
```

```
 */
xvobject image, position, printpixel;
static void infile_cb PROTO((xvobject, kaddr, kaddr));

void main(
     int  argc,
     char **argv,
     char **envp)
{
     kobject  object;
     char     *filename = "image:mandril";
     xvobject parent, infile;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

     /* create a data object from the information in the image file */
     object = kpds_open_input_object(filename);

        /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /* create a manager to contain printpixel & position objects */
     parent = xvw_create_manager(NULL, "parent");

     /*
      * create the image object below the inputfile object,
      * and associate it with the desired filen
      */
     image = xvw_create_image(parent, "image");
     xvw_set_attribute(image, XVW_IMAGE_IMAGEOBJ, object);

     /*
      * create the position object in the center below the image;
      * associate it with the same data using XVW_POSITION_OBJECT.
      */
     position = xvw_create_position(parent, "position");
     xvw_set_attributes(position,
                 XVW_BELOW,           image,
                 XVW_LEFT_OF,         image,
                 XVW_RIGHT_OF,        image,
                 XVW_POSITION_OBJECT, object,
                 NULL);

     /*
      * create an inputfile object so we can enter a new filename
      */
     infile = xvw_create_inputfile(parent, "infile");
     xvw_set_attributes(infile,
                         XVW_BELOW,              position,
                         XVW_TACK_EDGE,          KMANAGER_TACK_HORIZ,
                         XVW_INPUTFILE_FILENAME, filename,
                         NULL);
     xvw_add_callback(infile, XVW_INPUTFILE_CALLBACK, infile_cb, NULL);
```

```
     /*
      * create the printpixel object in an independent window,
         * associate it with the same data using XVW_PRINTPIXEL_OBJECT.
      * want 9 data pixel values in each row, 11 in each column.
      */
     parent = xvw_create_manager(NULL, "printpixel_parent");
     printpixel = xvw_create_printpixel(parent, "printpixel");
     xvw_set_attributes(printpixel,
                 XVW_PRINTPIXEL_OBJECT,    object,
                 XVW_PRINTPIXEL_WIDTH,     9,
                          XVW_PRINTPIXEL_HEIGHT,    11,
                 NULL);

     /* display & run the program */
     xvf_run_form();
}

/*
 *  callback to allow user to enter new filename
 */
static void infile_cb(
    xvobject object,
    kaddr    client_data,
    kaddr    call_data)
{
     char *filename;
     kobject data_object;

     xvw_get_attribute(object,     XVW_INPUTFILE_FILENAME, &filename);
     data_object = kpds_open_input_object(filename);

     xvw_set_attribute(image,      XVW_IMAGE_IMAGEFILE,     filename);
     xvw_set_attribute(position,   XVW_POSITION_FILENAME,   filename);
     xvw_set_attribute(printpixel, XVW_PRINTPIXEL_FILENAME, filename);
/*
        xvw_set_attribute(image,      XVW_IMAGE_IMAGEOBJ,    data_object);
        xvw_set_attribute(position,   XVW_POSITION_OBJECT,   data_object);
        xvw_set_attribute(printpixel, XVW_PRINTPIXEL_OBJECT, data_object);
*/

}
```

C.7. The Zoom Object

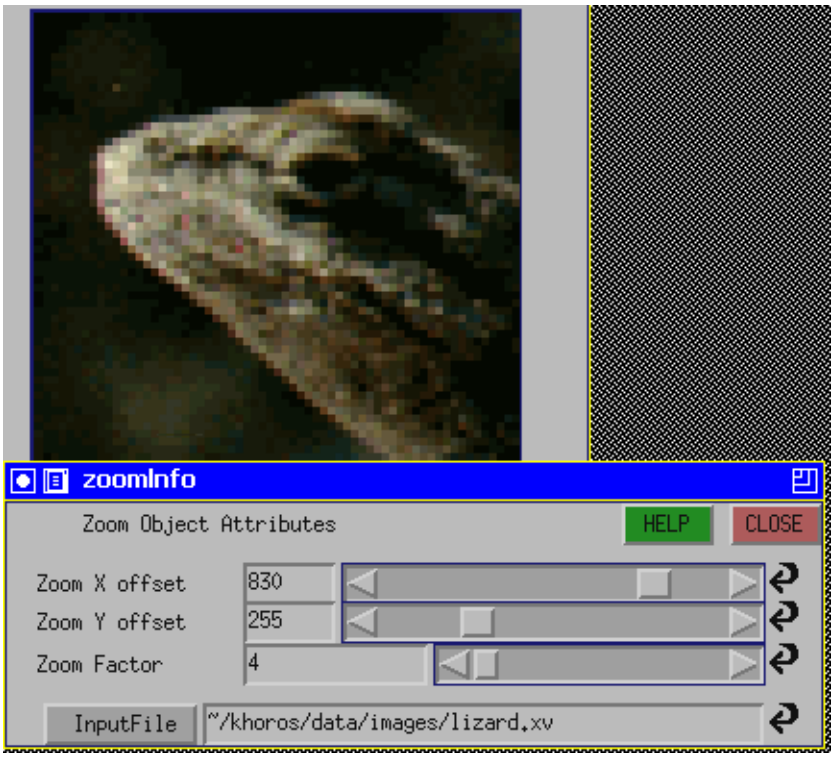


Figure 7: The zoom object with its internal menuform displayed. Here, the zoom object is used to zoom in on the "lizard.xv" image with a zoom factor of 4.0.

C.7.1. xvw_create_zoom() — *create a zoom object*

Synopsis

```
xvobject xvw_create_zoom(
    xvobject parent,
    char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the zoom object (for use in app-defaults files, etc)

Returns

The zoom object on success, NULL on failure

Description

The zoom object provides a zoom window which is used to zoom in on an image. It was designed for

use in conjunction with an image object.

The zoom factor can be set as desired. The zoom object may be set to update on either Button Press or Pointer Motion events.

C.7.2. Attributes of the Zoom Object

Summary of Zoom Attributes

Attribute	Description
XVW_ZOOM_FACTOR	This double value specifies the zoom factor, where values larger than one cause the zoom object to zoom in on the image, while values less than one cause the zoom object to zoom out on the image. For example, a value of 2.0 doubles the size of the image, while a value of 0.5 shrinks the image by half. The zoom factor is intentionally a double precision value; this means that in the case of pixel replication, pixels are rounded to their nearest location. This also implies that under certain circumstances, the zoom will not appear to stretch evenly over the entire image.
XVW_ZOOM_INTERPOLATE	Sets the type of interpolation. Currently, this can only be KZERO_ORDER, which is pixel replication.
XVW_ZOOM_LOCATIONMARKER	This attributes specifies the marker (cursor) which will be used to mark the center of the zoom window. Typically the zoom position should always be in the center of the zoom window, but if center image is FALSE, then the zoom object will reposition the position. In this case it is helpful to have an indicator that shows this position. The available marker types include a cross, a box, and a dot (or none).
XVW_ZOOM_UPDATEMODE	This attribute indicates the method with which the zoom window will update. Available settings include: which does continuous update as the pointer is moved across the image, and KZOOM_UM_BUTTON_PRESS, which updates only when the button is pressed at the desired location.
XVW_ZOOM_XPOSITION	This is the X position in the image at which the zoom window is focused; the zoom cursor in the zoom window will reflect this (x,y) position.
XVW_ZOOM_YPOSITION	This is the Y position in the image at which the zoom window is focused; the zoom cursor in the zoom window will reflect this (x,y) position.

Descriptions of Zoom Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_ZOOM_FACTOR (zoomFactor)	double	8.0	value > 0.0
XVW_ZOOM_INTERPOLATE (zoomInterpolate)	int	KZERO_ORDER	KZERO_ORDER
XVW_ZOOM_LOCATIONMARKER (zoomLocationmarker)	int	KZOOM_LM_BOX	KZOOM_LM_NONE KZOOM_LM_CROSS KZOOM_LM_BOX KZOOM_LM_CIRCLE KZOOM_LM_DOT
XVW_ZOOM_UPDATEMODE (zoomUpdatemode)	int	KZOOM_UM_CONTINUOUS	KZOOM_UM_CONTINUOUS KZOOM_UM_BUTTON_PRESS
XVW_ZOOM_XPOSITION (zoomXposition)	int	0	0 - image width
XVW_ZOOM_YPOSITION (zoomYposition)	int	0	0 - image height

C.7.3. Complete Resource Set of the Zoom Object

The inheritance tree of the zoom object is as follows:

manager -> graphics -> color -> image -> zoom

Accordingly, the complete resource set for the zoom object includes:

1. The zoom attribute resource set, given above
2. The image attribute resource set, given in section C.2, "Attributes of the Image Object"
3. The color attribute resource set, given in Section B, "The Color Attributes"
4. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
5. The general object attributes, given in Chapter 2, "The xvwidgets Library", Section B, "General Attributes of GUI and Visual Objects".

C.7.4. Example Using the Zoom Object

Examples of programs using the zoom object can be found in `$ENVISION/examples/zoom`. One of the simpler programs is as follows.

```
#include <envision.h>
```

```
/*
 * This program creates an image object which displays the ball image.
 * Then, a zoom object is created, and is associated with the same information
 * as the image object.  Without installing any event handlers, the zoom
 * object will automatically zoom in on the image displayed in the image object
 * when the pointer is moved across the image, since they are both using the
 * same data object.
 */

void main(
   int  argc,
   char *argv[])
{
     kobject  object;
     char     *filename = "image:ball";
     xvobject image, zoom, position, parent;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

     /* allow different images to be used, as in "% example image:lizard" */
     if (argc > 1)
        filename = argv[1];

     /* open the data object associated with the input image */
     object = kpds_open_input_object(filename);

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /*
      * a manager will be the backplane for the image and position objects
      */
     parent = xvw_create_manager(NULL, "parent");

     /*
      *  create the image object; associate it with the data object
         *  representing the input file
      */
     image  = xvw_create_image(parent, "image");
     xvw_set_attribute(image, XVW_IMAGE_IMAGEOBJ, object);

     /*
      *  create the position object under the image object; have it
         *  use the same data object.
      */
     position = xvw_create_position(parent, "position");
     xvw_set_attributes(position,
                  XVW_BELOW,           image,
                  XVW_POSITION_OBJECT, object,
                  XVW_LEFT_OF,         NULL,
                  XVW_RIGHT_OF,        NULL,
                  NULL);

     /*
```

```
         *  create the zoom object in an independant window.
            *  specify the associated image object to be the same data.
         */
        zoom = xvw_create_zoom(NULL, "zoom");
        xvw_set_attributes(zoom,
                    XVW_IMAGE_IMAGEOBJ,      object,
                    XVW_ZOOM_FACTOR,         5.0,
                    XVW_ZOOM_LOCATIONMARKER, KZOOM_LM_CROSS,
                    XVW_MINIMUM_WIDTH,       200,
                    XVW_MINIMUM_HEIGHT,      200,
                    NULL);


        /* display and run */
        xvf_run_form();
    }
```

D. Visual Objects Related to Colormaps

D.1. The ColorCell Object

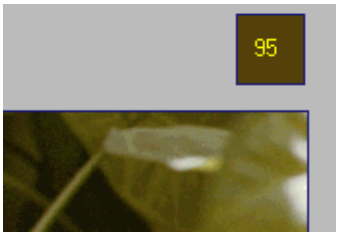


Figure 8: The colorcell object is used to display the color and pixel value of one or more pixels in an image.

D.1.1. xvw_create_colorcell() — *create a colorcell xvobject*

Synopsis

```
xvobject xvw_create_colorcell(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`

the parent object; NULL will cause a default toplevel to be created automatically

`name`
a name for this particular instance of the colorcell object (for use in app-defaults files, etc)

Returns

The colorcell object on success, NULL on failure

Description

A colorcell visual object is used to display the color and the value of pixels in an image object; thus, it is designed to be used in conjunction with an image object. It is comprised of a square area in which the color of the pixel(s) are displayed. When a single pixel is associated with the colorcell object, the numeric value of that pixel may also be shown. The colorcell object may be used simply to reflect the color of one or more pixels in the image, or it may be used to control those colors.

D.1.2. Attributes of the Colorcell Visual Object

Summary of ColorCell Attributes

Attribute	Description
XVW_COLORCELL_ADD	Use of this *write-only* attribute adds the specified image data value (pixel) to *colorcell list* (the list of pixels with which the colorcell is associated). If XVW_COLORCELL_UPDATE_ONADD is set to TRUE, use of this attribute will force the color in which a pixel value is displayed to be the same as the color currently displayed by the colorcell object.
XVW_COLORCELL_BLUEVAL	This attribute can be used to set the blue component of the color that is displayed by the colorcell; by extension, this will identically affect the color of the pixels in the image that are in the colorcell list.
XVW_COLORCELL_CLEAR	This *action* attribute clears the *colorcell list* (the list of pixels with which the colorcell is associated), so that the colorcell can subsequently be associated with new pixel value(s).
XVW_COLORCELL_DELETE	Use of this *write-only* attribute deletes the image data value (pixel) from the *colorcell list* (the list of pixels with which the colorcell is associated). If XVW_COLORCELL_RESTORE_ONDELETE is set to TRUE, use of this attribute will restore the pixel to its original color in which it appeared before it was added to the index list with XVW_COLOR-CELL_ADD.
XVW_COLORCELL_GREENVAL	This attribute can be used to set the green component of the color that is displayed by the colorcell; by extension, this will identically affect the color of the pixels in the image that are in the colorcell list.

Summary of ColorCell Attributes	
Attribute	**Description**
XVW_COLORCELL_INDEX	This attribute is the image data value (pixel) with which to associate the colorcell object. The colorcell object will display the color in which that pixel appears on the screen, and may display the pixel value as well.
XVW_COLORCELL_INDEXLIST	This *read-only* attribute allows you to obtain the array of image data values (pixels) currently associated with the colorcell object (ie, those pixels currently in the colorcell list). The array obtained will contain the pixel values which appear in the color reflected in the colorcell object.
XVW_COLORCELL_INDEXNUM	This is *read-only* attribute allows you to obtain the number of image data values (pixels) currently associated with the the colorcell object. Note that this is the size of the array returned by XVW_COLOR-CELL_INDEXLIST.
XVW_COLORCELL_REDVAL	This attribute can be used to set the red component of the color that is displayed by the colorcell; by extension, this will identically affect the color of the pixels in the image that are in the colorcell list.
XVW_COLORCELL_RESTORE	This *action* attribute restores the pixels in the *colorcell list* to the the original colors in which they appeared in before they added to the colorcell list using XVW_COLORCELL_ADD.
XVW_COLORCELL_RESTORE_ONDELETE	When this attribute is set to TRUE, the colorcell object will force pixels in the associated image to be immediately restored to their original color when they are deleted from the pixel list using XVW_COLOR-CELL_DELETE.
XVW_COLORCELL_SAVE_COLOR	Use of this *action attribute* saves the current color of the colorcell, whatever that may be, so that if the color is subsequently changed, and then restored, this will be the color that is restored. Note that this saved color will be restored if the XVW_COLORCELL_RESTORE action attribute is used, or if the XVW_COLORCELL_DELETE attribute is used to delete an index from the colorcell list when the XVW_COLOR-CELL_RESTORE_ONDELETE attribute is set to TRUE. Use of this action attribute is useful in making sure that restored colors will be correct if the any of the following attributes have been set on the object on which the colorcell has had the XVW_COLOR_COLOROBJ
XVW_COLORCELL_SHOWINDEX	When this attribute is set to TRUE, the colorcell object will display the pixel value with which it is associated; when it is set to FALSE, the colorcell object consists of a solid block of color. In general, when the colorcell object is associated with multiple pixel values, it makes more sense to set this attribute to FALSE; however, if it is set to TRUE, the colorcell widget will display the *first* pixel value in the pixel list.
XVW_COLORCELL_UPDATE	This *action attribute* updates the colors of the pixels in the colorcell list, according to the color currently displayed by the colorcell.

Summary of ColorCell Attributes	
Attribute	**Description**
XVW_COLORCELL_UPDATE_ONADD	When this attribute is set to TRUE, the colorcell object will force pixels in the associated image to immediately appear in the color displayed by the colorcell when they are added to the pixel list using XVW_COLOR-CELL_ADD.

Descriptions of ColorCellAttributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_COLORCELL_ADD (N/A)	int	N/A	valid pixel value of the displayed data object
XVW_COLORCELL_BLUEVAL (N/A)	int	0	0-255
XVW_COLORCELL_CLEAR (N/A)	int	N/A (action attribute)	TRUE
XVW_COLORCELL_DELETE (N/A)	int	N/A (action attribute)	valid pixel value of the displayed data object
XVW_COLORCELL_GREENVAL (N/A)	int	0	0-255
XVW_COLORCELL_INDEX (N/A)	int	none	any valid image pixel value
XVW_COLORCELL_INDEXLIST (N/A)	int *	N/A	integer array of image pixel values
XVW_COLORCELL_INDEXNUM (N/A)	int	N/A	size of the array specified by XVW_COLOR-CELL_INDEXLIST
XVW_COLORCELL_REDVAL (N/A)	int	0	0-255
XVW_COLORCELL_RESTORE (N/A)	int	N/A (action attribute)	TRUE
XVW_COLORCELL_RESTORE_ONDELETE (colorcellRestoreOndelete)	int	FALSE	TRUE/FALSE
XVW_COLORCELL_SAVE_COLOR (N/A)	int	N/A	TRUE
XVW_COLORCELL_SHOWINDEX (colorcellShowindex)	int	TRUE	TRUE/FALSE
XVW_COLORCELL_UPDATE (N/A)	int	N/A (action attribute)	TRUE
XVW_COLORCELL_UPDATE_ONADD (colorcellUpdateOnadd)	int	FALSE	TRUE/FALSE

D.1.3. Complete Resource Set of the ColorCell Visual Object

The inheritance tree of the colorcell object is as follows:

manager -> graphics -> color -> colorcell

Accordingly, the complete resource set for the colorcell object includes:

1. The colorcell attribute resource set, given above
2. The color attribute resource set, given in Section B, "The Color Attributes"
3. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
4. The general object attributes, given in Chapter 2, "The xvwidgets Library", Section B, "General Attributes of GUI and Visual Objects".

D.1.4. Example using the ColorCell Visual Object

Example programs using the colorcell visual object can be found in `$ENVISION/examples/color/colorcell`. The simplest of these is as follows.

```
#include <envision.h>

void  update_colorcell   PROTO((xvobject, kaddr, XEvent *, int *));

/*
 *  This program creates an image object which displays the kitten image.
 *  Then, a colorcell object is created. A colorcell object is associated with
 *  a pixel value in an image; it is a box filled with the color in which
 *  the pixel appears, and labelled with the pixel value.
 *
 *  An event handler is installed which allows you to click the mouse in
 *  the image;  the colorcell object will be updated with the color and
 *  value of the pixel on which the mouse was clicked.
 */
void main(
   int  argc,
   char *argv[])

{
     xvobject manager, image, colorcell, label;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }
```

```
     /* Create a manager to parent the image and colorcell objects  */
     manager = xvw_create_manager(NULL, "manager");

     /* create a 35x35 colorcell object in the  upper right hand corner */
     colorcell = xvw_create_colorcell(manager, "colorcell");
     xvw_set_attributes(colorcell,
          XVW_COLOR_COLORFILE, "image:kitten",
          XVW_BELOW,     NULL,
          XVW_WIDTH,     35,
          XVW_HEIGHT,          35,
          XVW_LEFT_OF,         NULL,
          NULL);

     /* put a label, just to be fancy */
     label = xvw_create_label(manager, "label");
     xvw_set_attributes(label,
                  XVW_LEFT_OF, colorcell,
                  XVW_ABOVE,   colorcell,
                  XVW_BELOW,   colorcell,
                  XVW_LABEL,   "This is the colorcell object =>",
                     NULL);

     /* create the image object below the colorcell; specify the image */
     image = xvw_create_image(manager, "image");
     xvw_set_attributes(image,
          XVW_IMAGE_IMAGEFILE, "image:kitten",
          XVW_BELOW,     colorcell,
          NULL);

     /* add the event handler to update the colorcell */
     xvw_add_event(image, ButtonPressMask | ButtonMotionMask,
                     update_colorcell, (kaddr)colorcell);

     /* display & run */
     xvf_run_form();
}


/*
 *  event handler to update the colorcell index
 */
void  update_colorcell(
    xvobject object,
    kaddr    clientData,
    XEvent   *event,
    int  *dispatch)
{
     double   value;
     xvobject colorcell = (xvobject) clientData;

     /* get the value of the pixel where the mouse was clicked */
     xvw_get_attribute(object, XVW_IMAGE_VALUE, &value);

     /* update the colorcell to reflect that pixel value */
     xvw_set_attribute(colorcell, XVW_COLORCELL_INDEX, (int) value);
}
```

D.2. The Palette Object

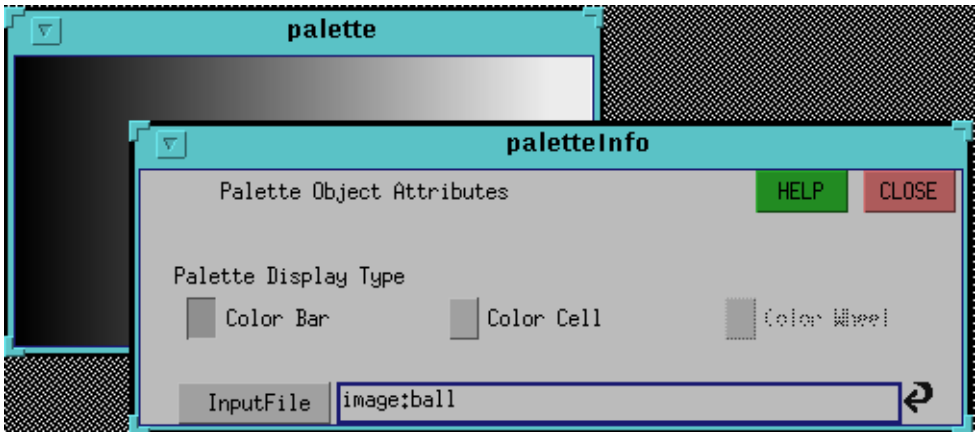


Figure 9: The palette object with its internal menuform displayed. Here, the palette object is used to display the color palette for the "ball.xv" using a the rectangular palette display type.

D.2.1. xvw_create_palette() — *create a palette object*

Synopsis

```
xvobject xvw_create_palette(
    xvobject parent,
    char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the palette object (for use in app-defaults files, etc)

Returns

The palette xvobject on success, NULL otherwise

Description

The palette visual object provides a visual display of the colors defined by a colormap. The colors may be displayed as a grid of color cells, a color palette, or a color wheel.

Note that when an RGB image is displayed on an 8-bit screen, a 3-3-2 transformation is used in order to convert the RGB image into an 8-bit representation. Because of this, the colors in the palette object that are displayed in this situation will reflect the 3-3-2 map, and may not seem to correspond to the

displayed image.

D.2.2. Attributes of the Palette Visual Object

Summary of Palette Attributes

Attribute	Description
XVW_PALETTE_ADD	Use of this *write-only* attribute adds the specified image data value (pixel) to *palette list* (the list of pixels highlighted in the palette).

XVW_PALETTE_CALLBACK	If desired, *xvw_add_callback()* or *xvw_insert_callback()*may be used to install a callback on the palette object which will be fired when the user selects a color cell from the palette object. When calling *xvw_add_callback()*, pass this attribute directly, as in
	
	`xvw_add_callback(palette_object, XVW_PALETTE_CALLBACK,`
	`palette_callback, client_data);`
	The index of the colorcell that was selected by the user is passed into the callback as the call_data; this integer *pointer* must be cast correctly before use. It is the responsibility of the palette callback to keep a list of the color cells that have been selected (if this information will be needed). The callback must also set the attributes XVW_PALETTE_ADD to select (stipple) a color cell and XVW_PALETTE_DELETE to un-select (un-stipple) a color cell. The following code segment gives an example of a callback that might be installed on a palette object:
	
	`static void test_callback(`
	`xvobject palette,`
	`kaddr client_data,`
	`kaddr call_data)`
	`{`
	`static int *indexes = NULL;`
	`static int index_num = 0;`
	
	`int *indx = (int *) call_data;`
	
	`if (karray_locate(indexes, KINT, *indx, index_num) == -1)`
	`{`
	`*indx);`
	`*indx);`
	`}`

XVW_PALETTE_CLEAR	This *action* attribute clears the *palette list* so that no pixels are highlighted in the palette.

Summary of Palette Attributes	
Attribute	**Description**
XVW_PALETTE_DELETE	Use of this *write-only* attribute deletes the image data value (pixel) from the *palette list* (the list of pixels highlighted in the palette).
XVW_PALETTE_INDEXLIST	This *read-only* attribute allows you to obtain the palette list (ie, those data values (pixels) in the palette that are currently highlighted). By default, there is no palette list (ie, no pixels in the palette are highlighted).
XVW_PALETTE_INDEXNUM	This is *read-only* attribute allows you to obtain the number of image data values (pixels) currently highlighted in the palette object. Note that this is the size of the array returned by XVW_PALETTE_INDEXLIST.
XVW_PALETTE_TYPE	This attribute specifies how the palette will be displayed; it may be one of: PALETTE_TYPE_COLORBAR ("Color Bar" on the menuform) for a linearly increasing color bar. KPALETTE_TYPE_COLORCELL ("Color Cell" on the menuform) for a palette of rectangular color cells. or PALETTE_TYPE_COLORWHEEL ("Color Wheel" on the menuform) for a pie chart of colors.

Descriptions of Palette Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_PALETTE_ADD (N/A)	int	N/A	valid pixel value of the displayed data object
XVW_PALETTE_CALLBACK (N/A)	void (*call-back_rou-tine)(xvob-ject, kaddr, kaddr)	NULL	callback function, in the form: void palette_callback xvobject object, kaddr client_data, kaddr call_data)
XVW_PALETTE_CLEAR (N/A)	int	N/A (action attribute)	TRUE
XVW_PALETTE_DELETE (N/A)	int	N/A (action attribute)	valid pixel value of the displayed data object
XVW_PALETTE_INDEXLIST (N/A)	int *	N/A	integer array of image pixel values
XVW_PALETTE_INDEXNUM (N/A)	int	N/A	size of the array specified by XVW_PALETTE_INDEXLIST
XVW_PALETTE_TYPE (paletteType)	int	KPALETTE_TYPE_COLORBAR	KPALETTE_TYPE_COLORBAR KPALETTE_TYPE_COLORCELL KPALETTE_TYPE_COLORWHEEL

D.2.3. Complete Resource Set of the Palette Visual Object

The inheritance tree of the palette object is as follows:

manager -> graphics -> color -> palette

Accordingly, the complete resource set for the palette object includes:

1. The palette attribute resource set, given above
2. The color attribute resource set, given in Section B, "The Color Attributes"
3. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
4. The general object attributes, given in Chapter 2, "The xvwidgets Library", Section B, "General Attributes of GUI and Visual Objects".

D.2.4. Example Using the Palette Visual Object

Examples of programs using the 2D Plot object can be found in `$ENVISION/examples/color/palette/`. One of these is as follows.

```
#include <envision.h>

/*
 *  This example program creates a palette object.
 *  A palette object displays a rectangular area in which each pixel
 *  value from the image appears in the color that is displayed on the image.
 *  Pixel values are sorted in increasing order starting at zero.
 *
 *  A callback is used to keep track of which pixels are selected by
 *  the user from the palette object.
 */
static void test_callback PROTO((xvobject, kaddr, kaddr));

void main(
   int  argc,
   char *argv[])
{
     kobject  object;
     xvobject parent, palette;
     char *filename = "image:lizard";

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }
```

```
    /* get the data object associated with the input image */
    object = kpds_open_input_object(filename);

    /*
        *  create the palette object, and associate it with the data object
        *  representing the input file
     */
    parent  = xvw_create_manager(NULL, "parent");
    xvw_set_attributes(parent,
                        XVW_WIDTH,          375,
                        XVW_HEIGHT,         275,
                        NULL);

    palette = xvw_create_palette(parent, "palette");
    xvw_set_attributes(palette,
                XVW_COLOR_COLOROBJ, object,
                XVW_TACK_EDGE,      KMANAGER_TACK_ALL,
                XVW_PALETTE_TYPE,   KPALETTE_TYPE_COLORCELL,
                   NULL);

    xvw_add_callback(palette, XVW_PALETTE_CALLBACK, test_callback, NULL);

    /* display and run */
    xvf_run_form();
}

/*
 *  This test_callback is fired when the user clicks on a color cell of the
 *  palette object.  It keeps an internal list of color cell indexes that
 *  have been selected by the user (for whatever use we may have for them,
 *  we are not doing anything with them here).  It sets the XVW_PALETTE_ADD
 *  attribute for each color cell that is selected; that's what makes it
 *  appear stippled.  It sets the XVW_PALETTE_DELETE attribute when the
 *  color cell is selected again; that's what puts it back to "normal".
 *  Also note that you can call the XVW_PALETTE_CLEAR attribute to clear
 *  (unselect) all the cells in the palette; when using that one, you
 *  should karray_free(indexes, KINT, index_num, NULL) and reset indexes=NULL.
 *
 */
static void test_callback(
   xvobject palette,
   kaddr    client_data,
   kaddr    call_data)
{
    static int *indexes = NULL;
    static int index_num = 0;

    int *indx = (int *) call_data;

    if (karray_locate(indexes, KINT, *indx, index_num) == -1)
    {
        xvw_set_attribute(palette, XVW_PALETTE_ADD, *indx);
        kfprintf(kstderr, "adding cell %d to list of selected cells\n",
              *indx);
        indexes = karray_add(indexes, KINT, *indx, index_num++);

    }
    else
    {
```

```
                xvw_set_attribute(palette, XVW_PALETTE_DELETE, *indx);
                kfprintf(kstderr, "deleting cell %d from list of selected cells\n",
                        *indx);
                indexes = karray_delete(indexes, KINT, *indx, index_num--);

        }
}
```

D.3. The PrintMapVal Object

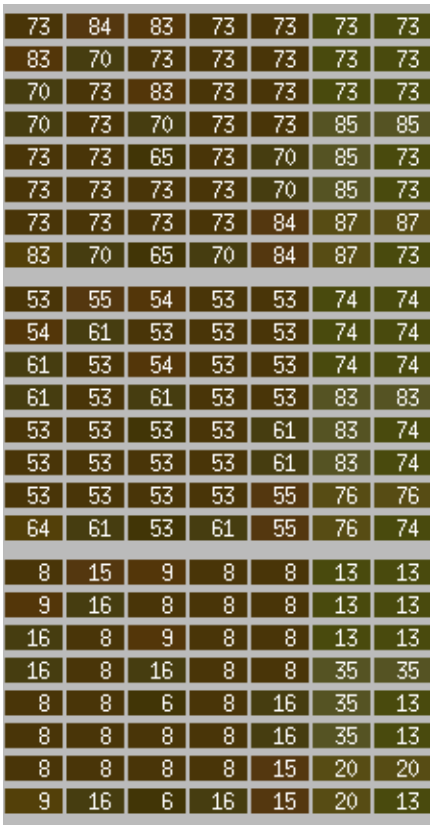


Figure 10: A printmapval object prints the colormap values that define the color of each pixel in an image as the pointer is moved over the image. The printmapval object is similar to the printpixel object, except that the values printed originate in the colormap, not in the image data, and there are typically three displays, one each for the Red, Green, and Blue map columns of the colormap.

D.3.1. xvw_create_printmapval() — *create a printmapval xvobject*

Synopsis

```
xvobject xvw_create_printmapval(
```

```
xvobject parent,
char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the printmapval object (for use in app-defaults files, etc)

Returns

The printmapval xvobject on success, NULL otherwise

Description

A printmapval object provides a mechanism which will track pointer movement in an image object and print the values of the map data indexed by the pixel under the pointer. It was designed for use in conjunction with an image object, and is only useful with data objects having a map segment.

The printmapval object is made up of one or more (width x height) sets of label objects; typically, there will be three such sets, one each for the Red, Green, and Blue columns of the colormap. The background of each label is the color defined by the values of the map columns at that pixel, and the value displayed is the value of the corresponding map column.

D.3.2. Attributes of the PrintMapVal Object

Summary of PrintMapVal Attributes

Attribute	Description
`XVW_PRINTMAPVAL_CLIPFILE`	The name of a file containing a clip mask may be specified using this attribute; the default is NULL. Note that this attribute is mutually exclusive with `XVW_PRINTPIXEL_CLIPOBJ`; use one or the other, not both. See the explanation of `XVW_PRINTPIXEL_CLIPOBJ` for more details on the clip mask.

Summary of PrintMapVal Attributes	
Attribute	**Description**
XVW_PRINTMAPVAL_CLIPOBJ	A clip mask is used to obscure portions of the displayed map values; if desired, this data object (kobject) attribute can be used to set a clip mask. When a clip mask is used with the printmapval object, the only map values that will be displayed by the printmapval object will be those that correspond to a value data of (1) in the clip mask. Map values corresponding to a value of (0) in the clip mask will not be displayed. Note that ONLY images of data type bit can be used as clip masks; if the input image to be used as a clip mask is not of type bit, it will be internally converted to type bit, without prompting or warning, before it is used. Note that this conversion will not affect the original file in any way. If a clip mask currently being used is no longer desired, set this attribute back to NULL. Note that this attribute is mutually exclusive with XVW_PRINTMAPVAL_CLIPFILE; use one or the other, not both.
XVW_PRINTMAPVAL_FILENAME	The file containing the data object for which the map values are to be displayed. Note that this attribute is mutually exclusive with XVW_PRINTMAPVAL_OBJECT; use one or the other, not both.
XVW_PRINTMAPVAL_HEIGHT	Specifies the number of map column values that should be displayed in the vertical direction on the printmapval display.
XVW_PRINTMAPVAL_OBJECT	This is the data object (kobject) containing the image for which map values are to be displayed. Note that this attribute is mutually exclusive with XVW_PRINTMAPVAL_FILENAME; use one or the other, not both.
XVW_PRINTMAPVAL_POLICY	This attribute indicates whether the printmapval object is to display the values from the colormap that is being used to display the data object (ie, the RGB values that may have been converted and/or normalized), or the values from the actual map segment of the data object being displayed (ie, the map values before they are converted and/or normalized for use in defining the colors that appear on the screen). Values include: KPRINTMAPVAL_DISPLAYEDVALUES ("Values Associated With Displayed Color" on the menuform) KPRINTMAPVAL_MAPDATAVALUES ("Actual Values from Map Data" on the menuform)
XVW_PRINTMAPVAL_SHOWCOLOR	This attribute indicates whether or not the color of the pixel under the pointer should be displayed as the background color for the labelstring object in which the map values are displayed. When set to FALSE, the background will be black.
XVW_PRINTMAPVAL_UPDATEMODE	This attribute indicates whether the printmapval display is to be updated continuously as the pointer moves across the image, or not until the button is clicked at a particular location in the image. Values include: KPRINTMAPVAL_UM_CONTINUOUS, ("Continuous" on the menuform), or KPRINTMAPVAL_UM_BUTTONPRESS ("ButtonPress" on the menuform)
XVW_PRINTMAPVAL_WIDTH	Specifies the number of map column values that should be displayed in the horizontal direction on the printmapval display.

Summary of PrintMapVal Attributes	
Attribute	**Description**
XVW_PRINTMAPVAL_XPOSITION	This is the X position in the image corresponding to the map value that appears in the upper left hand corner of the printmapval grid.
XVW_PRINTMAPVAL_YPOSITION	This is the Y position in the image corresponding to the map value that appears in the upper left hand corner of the printmapval grid.

Descriptions of PrintMapVal Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_PRINTMAPVAL_CLIPFILE (N/A)	char *	NULL	valid name of input file containing clip mask
XVW_PRINTMAPVAL_CLIPOBJ (N/A)	kobject	NULL	valid data object defining clip maks
XVW_PRINTMAPVAL_FILENAME (N/A)	char *	NULL	valid input filename
XVW_PRINTMAPVAL_HEIGHT (printmapvalHeight)	int	9	1 <= height <= image height
XVW_PRINTMAPVAL_OBJECT (N/A)	kobject	NULL	valid data object
XVW_PRINTMAPVAL_POLICY (printmapvalPolicy)	int	KPRINTMAPVAL_DISPLAYEDVALUES	KPRINTMAPVAL_DISPLAYEDVALUES KPRINTMAPVAL_MAPDATAVALUES
XVW_PRINTMAPVAL_SHOWCOLOR (printmapvalShowcolor)	int	FALSE	TRUE/FALSE
XVW_PRINTMAPVAL_UPDATEMODE (printmapvalUpdatemode)	int	KPRINTMAPVAL_UM_CONTINUOUS	KPRINTMAPVAL_UM_CONTINUOUS KPRINTMAPVAL_UM_BUTTONPRESS
XVW_PRINTMAPVAL_WIDTH (printmapvalWidth)	int	7	1 <= width <= image width
XVW_PRINTMAPVAL_XPOSITION (N/A)	int	0	0 - image width
XVW_PRINTMAPVAL_YPOSITION (N/A)	int	0	0 - image height

D.3.3. Resource Set of the PrintMapVal Object

The inheritance tree of the printmapval object is as follows:

manager -> graphics -> color -> printmapval

Accordingly, the complete set of attributes for the printmapval object includes:

1. The printmapval object attribute resource set, given above

2. The color attribute resource set, given in Section B, "The Color Attributes"

3. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"

4. The general object attributes, given in Chapter 2, "The xvwidgets Library", Section B, "General Attributes of GUI and Visual Objects".

D.3.4. Example Using the PrintMapVal Visual Object

Examples using the printmapval visual object can be found in `$ENVISION/examples/color/printmapval/`. The simplest of these is as follows:

```
#include <envision.h>

/*
 *  This program creates an image object, a position object to reflect the
 *  location of the mouse pointer in the image, and a printmapval object to
 *  print the values of the map values indexed by the pixels surrounding that
 *  location.
 *
 *  Note that you do NOT have to write an event handler to make the
 *  printmapval object update; the fact that we have created an image object
 *  using the same file for its data does the trick.  The image object will
 *  recognise pointer motion within it, and cause the printmapval object (and
 *  the position object as well) to be updated automatically.
 */
static void change_policy PROTO((xvobject, kaddr, kaddr));

void main(
     int  argc,
     char **argv,
     char **envp)
{
     kobject  object;
     char     *filename = "image:mandril-rgb";
     xvobject image, position, parent;
     xvobject label, button, printmapval;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

     /* create a data object from the information in the image file */
     object = kpds_open_input_object(filename);

        /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /* create a manager to contain image & position objects */
     parent = xvw_create_manager(NULL, "image_parent");

     /* create the image object, associate it with the desired file */
```

```
    image = xvw_create_image(parent, "image");
    xvw_set_attribute(image, XVW_IMAGE_IMAGEOBJ, object);

    /*
     * create the position object in the center below the image;
     * associate it with the same data using XVW_POSITION_OBJECT.
     */
    position = xvw_create_position(parent, "position");
    xvw_set_attributes(position,
                XVW_BELOW,           image,
                XVW_LEFT_OF,         NULL,
                XVW_RIGHT_OF,        NULL,
                XVW_POSITION_OBJECT, object,
                NULL);

    /* create a manager to contain image & position objects */
    parent = xvw_create_manager(NULL, "printmapval_parent");

    /* create printmapval object */
    label = xvw_create_label(parent, "label");
    xvw_set_attributes(label,
                XVW_RIGHT_OF, NULL,
                XVW_LABEL,    "PrintMapVal Policy:",
                NULL);

    button = xvw_create_button(parent, "button");
    xvw_set_attributes(button,
                XVW_RIGHT_OF, label,
                XVW_LABEL,    "Print Displayed Map Values",
                NULL);

    printmapval = xvw_create_printmapval(parent, "printmapval");
    xvw_set_attributes(printmapval,
                XVW_BELOW,                 button,
                XVW_PRINTMAPVAL_OBJECT,    object,

                XVW_PRINTMAPVAL_WIDTH,     9,
                        XVW_PRINTMAPVAL_HEIGHT,    9,

                XVW_PRINTMAPVAL_SHOWCOLOR, TRUE,
                NULL);


    xvw_add_callback(button, XVW_BUTTON_SELECT,
               change_policy, printmapval);

    /* display & run the program */
    xvf_run_form();
}

static void change_policy(
    xvobject object,
    kaddr    client_data,
    kaddr    call_data)
{
    int policy;
    xvobject printmapval = (xvobject) client_data;

    xvw_get_attribute(printmapval, XVW_PRINTMAPVAL_POLICY, &policy);
```

```
        if (policy == KPRINTMAPVAL_MAPDATAVALUES)
        {
            xvw_set_attribute(printmapval, XVW_PRINTMAPVAL_POLICY,
                        KPRINTMAPVAL_DISPLAYEDVALUES);
            xvw_set_attribute(object, XVW_LABEL, "Print Displayed Map Values");
        }
        else
        {
            xvw_set_attribute(printmapval, XVW_PRINTMAPVAL_POLICY,
                        KPRINTMAPVAL_MAPDATAVALUES);
            xvw_set_attribute(object, XVW_LABEL, "Print Actual  Map Values");
        }
    }
```

D.4. The PseudoColor Object

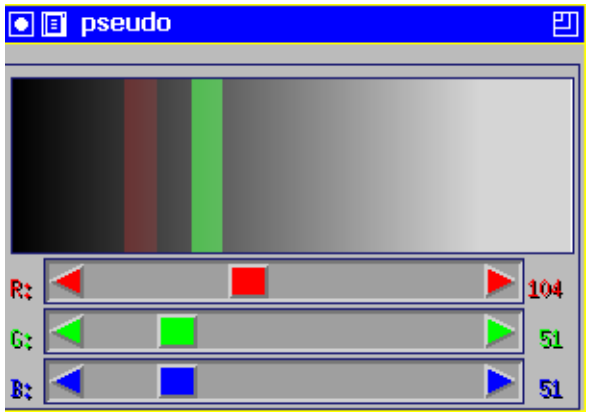


Figure 11: The pseudocolor object provides a method of changing the colormap of another object. Here, the RGB scrollbars are used to change the color associated with a range of pixel values.

D.4.1. xvw_create_pseudo() — *create a pseudo xvobject*

Synopsis

```
xvobject
xvw_create_pseudo(
    xvobject parent,
    char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`

a name for this particular instance of the pseudo object (for use in app-defaults files, etc)

Returns

The pseudocolor object on success, NULL otherwise

Description

The pseudocolor visual object provides a method of changing the colormap of another visual object. It displays the colors associated with the pixel values of the displayed object (as a color palette, colorcell grid, or color wheel). Under the color display is a set of three scrollbars, representing * Red, Green, and Blue.

Select pixels for color change from the color palette, colorcell grid, or color wheel. Next, move the thumbs of the three scrollbars underneath to change the red, green, and blue values defining the color in which the selected pixels appear. If desired, specific values for red, green, and blue may be typed directly into the text parameter boxes that appear to the right of the scrollbars. A "Reset" button at the top of the pseudocolor object provides a convenient way to un-select previously selected pixels.

The pixels to be changed are selected from the colorcell grid (or color palette, or color wheel). Pixels may be selected for color change using the following methods. These descriptions assume use of the colorcell grid, but pixel selection for the color palette and color wheel work similarly. Pixels may be selected individually, or in ranges. Note that *all* selected pixels on the colorcell grid will have their color changed when you set the red, green, and blue values. The "Reset" button un-selects *all* selected pixels in the colorcell grid.

1) To select a single pixel for color change, click on the color cell representing the desired pixel. The color cell will appear intented and stippled when it is selected. To unselect the pixel, click the mouse button on it again, or use the "Reset" button.

2) To select a range of pixels for color change, click on a pixel at the end of the range to be changed. Holding the button down, move the mouse to the desired end point; the range may be moved back and forth as long as the mouse button is held down. Releasing the mouse button sets the range. Multiple ranges may be selected by repeating this process on previously unselected pixels. Portions of a selected range may be unselected by repeating on previously selected pixels. To unselect the range, repeat the selection process, or use the "Reset" button.

The `XVW_COLOR_COLOROBJ` attribute is used to specify the data object whose colormap the pseudo-color object will be used to modify.

Note that pseudocolor should not be done on RGB images when the calling application is displayed on an 8-bit screen. This is because a mapping transformation is used in order to convert the RGB image into an 8-bit representation for visual display on an 8-bit screen.

Because of this transformation, pseudocoloring an RGB image on an 8-bit screen will not make sense; any changes to the colormap made via thresholding would alter the 24-bit to 8-bit transformation (a 3-3-2 RGB map is used), rather than the RGB values themselves, producing an unexpected result.

D.4.2. Attributes of the PseudoColor Visual Object

Summary of Pseudo Attributes	
Attribute	**Description**
XVW_PSEUDO_ADD	This *action attribute* adds the pixel value specified to the pseudocolor list of pixels. This can have two effects, depending on whether there are already pixels in the pseudocolor list. (1) if *no* pixels are currently in the pseudocolor list, XVW_PSEUDO_ADD will cause the pseudocolor object's RGB scrollbars to reflect the RGB components of color of the pixel specified. (2) if there are *already* one or more pixels in the pseudocolor list, XVW_PSEUDO_ADD will cause the pixel being added to the list to change to the color currently reflected by the RGB scrollars of the pseudocolor object; pixels currently on the pseudocolor list will already be displayed in this color. In either case, subsequent movement of the pseudocolor RGB scrollbars will change the color in which the pixel is displayed to the color specified by the scrollbars. Note that if desired, the XVW_IMAGE_VALUE attribute can be used to obtain the pixel value over which the pointer is positioned; the XVW_PSEUDO_ADD attribute may then be set to that pixel value. Note that if desired, the XVW_PSEUDO_CLEAR action attribute may be used to clear the psuedocolor list prior to adding a new pixel value, thus achieving the first effect.
XVW_PSEUDO_ALPHAVAL	This is the alpha channel value that determines the blending factor that is applied to the color defined by the Red, Green, and Blue values. A value of 1.0 for the alpha channel implies that the color defined by the RGB values is completely "solid" (ie, the only color showing is that which is defined by the RGB values; none of the background color will "show through"). A value of 0.0 for the alpha channel implies that the color defined by the RGB values is completely "transparent" (ie, the only color showing is that of the background; none of the color defined by the RGB values will be displayed). As the alpha channel value moves from 0.0 to 1.0, the color defined by the RGB values will become less "transparent" and more "solid"; the background color will "show though" less and less, until it does not appear at all.
XVW_PSEUDO_BLUEVAL	This is the blue component of the color in which the pixel value(s) that are associated with the pseudocolor object appear.

Summary of Pseudo Attributes	
Attribute	**Description**

XVW_PSEUDO_CALLBACK	If desired, a callback may be installed on the pseudocolor object that will be fired each time the user employs one of the RGB scrollbars to change the colormap of the displayed object. An *xvw_pseudo_struct* will be passed in as the call_data; cast this parameter accordingly before using, as in:
	
	`xvw_pseudo_struct *pseudo_struct;`
	`pseudo_struct = (xvw_pseudo_struct *) call_data;`
	The *xvw_psuedo_struct* is defined in $ENVISION/include/xvimage/Psuedo.h as follows:
	
	`typedef struct`
	`{`
	`xvobject object;`
	`int value;`
	`int type;`
	`} xvw_pseudo_struct;`

XVW_PSEUDO_CLEAR	This *action attribute* clears all pixel(s) from the pseudocolor list.
XVW_PSEUDO_DELETE	This *action attribute* deletes the pixel value specified to the pseudocolor list of pixels. Subsequent movement of the pseudocolor RGB scrollbars will have no effect on the color of the pixel once it is deleted from the psuedocolor list.
XVW_PSEUDO_GREENVAL	This is the green component of the color in which the pixel value(s) that are associated with the pseudocolor object appear.
XVW_PSEUDO_INDEX1	If the pseudocolor scrollbars are to operate on a range of pixels, this is the first pixel value (colormap index) in the range.
XVW_PSEUDO_INDEX2	If the pseudocolor scrollbars are to operate on a range of pixels, this is the last pixel value (colormap index) in the range.
XVW_PSEUDO_INDEXLIST	This *read-only* attribute may be used to obtain an array of integers representing the pixel values (colormap indices) that are currently associated with the pseudocolor object; ie, those pixels that will have their color changed when the user moves the RGB scrollbars.
XVW_PSEUDO_INDEXNUM	This is *read-only* attribute may be used to obtain the size of the integer array that is returned by the XVW_PSEUDO_INDEXLIST attribute.
XVW_PSEUDO_PALETTE_OBJECT	This *read_onlyP attribute is the palette component of the pseudo object. The palette component is where the actual pixels are displayed and can be interactively selected.*
XVW_PSEUDO_PALETTE_TYPE	The pseudocolor palette may have a type of Color Bar, Color Cell, or Color Wheel. Color bar is a linearly increasing colorbar palette. Color cell is the palette of square color cells that is used by default. Color wheel presents the palette as a pie chart of colors.
XVW_PSEUDO_REDVAL	This is the red component of the color in which the pixel value(s) that are associated with the pseudocolor object appear.

Summary of Pseudo Attributes	
Attribute	**Description**
XVW_PSEUDO_SHOW_PALETTE	This attribute causes the palette of linear colormap values to be displayed above the RGB scrollbars. Set to FALSE if the palette is not to be shown.
XVW_PSEUDO_UPDATE_ONADD	When set to TRUE, this attribute will cause pixels to immediately change to the color specified by the pseudo object's RGB scrollbars when they are added to the pseudocolor list with XVW_PSEUDO_ADD When set to FALSE, pixels do not change color until the user changes the color specified by the RGB scrollbars.
XVW_PSEUDO_USE_ALPHA	TRUE if an integer object is to be created so that the user can change the alpha channel; FALSE otherwise.

Descriptions of Pseudo Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_PSEUDO_ADD (N/A)	int	N/A	pixel values that appear in the data of the displayed image
XVW_PSEUDO_ALPHAVAL (N/A)	float	0.0	0.0 - 1.0
XVW_PSEUDO_BLUEVAL (N/A)	int	0	0-255
XVW_PSEUDO_CALLBACK (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_PSEUDO_CLEAR (N/A)	int	N/A	TRUE (action attribute)
XVW_PSEUDO_DELETE (N/A)	int	N/A	pixel values currently in the pseudocolor list, having been added previously with the use of XVW_PSEUDO_ADD
XVW_PSEUDO_GREENVAL (N/A)	int	0	0-255
XVW_PSEUDO_INDEX1 (N/A)	int	N/A	valid colormap index
XVW_PSEUDO_INDEX2 (N/A)	int	N/A	valid colormap index
XVW_PSEUDO_INDEXLIST (N/A)	int *	NULL	integer array containing index list
XVW_PSEUDO_INDEXNUM (N/A)	int	0	size of index list array given by XVW_PSEUDO_INDEXLIST

Descriptions of Pseudo Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_PSEUDO_PALETTE_OBJECT (N/A)	xvobject	NULL	valid xvobject
XVW_PSEUDO_PALETTE_TYPE (pseudoPaletteObject.paletteType)	int	KPALETTE_TYPE_COLORBAR	KPALETTE_TYPE_COLORBAR KPALETTE_TYPE_COLORCELL KPALETTE_TYPE_COLORWHEEL
XVW_PSEUDO_REDVAL (N/A)	int	0	0-255
XVW_PSEUDO_SHOW_PALETTE (pseudoShowPalette)	int	TRUE	TRUE/FALSE
XVW_PSEUDO_UPDATE_ONADD (pseudoUpdateOnadd)	int	FALSE	TRUE/FALSE
XVW_PSEUDO_USE_ALPHA (pseudoUseAlpha)	int	FALSE	TRUE/FALSE

D.4.3. Complete Resource Set of the PseudoColor Visual Object

The inheritance tree of the pseudocolor object is as follows:

manager -> graphics -> color -> pseudo

Accordingly, the complete resource set for the pseudocolor object includes:

1. The psuedocolor attribute resource set, given above
2. The color attribute resource set, given in Section B, "The Color Attributes"
3. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
4. The general object attributes, given in Chapter 2, "The xvwidgets Library", Section B, "General Attributes of GUI and Visual Objects".

D.4.4. Example Using the Pseudocolor Visual Object

There are a number of programs using the pseudocolor object; they may be found in `$ENVISION/examples/color/pseudo/`. The simplest of these is as follows:

```
#include <envision.h>

/*
 *  This example simply displays a psuedocolor object working off the
 *  image information found in the "image:ball" image file.  A pseudocolor
 *  object displays a set of the colorcells found in the image; below the
```

```
 *  palette are three scrollbars to control the Red, Green, and Blue elements
 *  of a particular color.  Select a "pseudocolor range" by clicking the
 *  first mouse button on two colorcells in the palette; then, use the Red,
 *  Green, and Blue scrollbars to change the color displayed by the pixels in
 *  that range.
 */
void main(
  int argc,
  char **argv,
  char **envp)
{
     kobject  ball;
        xvobject palette, pseudo, parent;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

     /* initialize the xvwidgets lib */
        if (!xvw_initialize(XVW_MENUS_XVFORMS))
        {
           kerror(NULL, "main", "unable to open display");
           kexit(KEXIT_FAILURE);
        }

     /* create a pseudocolor object w/ image:ball as target image */
        parent = xvw_create_manager(NULL, "manager");
     xvw_set_attributes(parent,
                    XVW_WIDTH,  400,
                    XVW_HEIGHT, 250,
                 NULL);
        pseudo = xvw_create_pseudo(parent, "pseudo");
     xvw_set_attributes(pseudo,
                 XVW_TACK_EDGE, KMANAGER_TACK_ALL,
                         XVW_LEFT_OF,   NULL,
                         XVW_RIGHT_OF,  NULL,
                         XVW_BELOW,     NULL,
                         XVW_ABOVE,     NULL,
                         NULL);

     xvw_activate_menu(pseudo);


     /* open data object associated with ball informatin */
        ball = kpds_open_input_object("image:ball");

     /*
      * use XVW_COLOR_COLOROBJ to associate the pseudocolor object with
          * the ball data object.  alternatively, we could have used
          * XVW_COLOR_COLORFILE to specify the "image:ball" file directly
          */
        xvw_get_attribute(pseudo, XVW_PSEUDO_PALETTE_OBJECT, &palette);
        xvw_set_attribute(palette, XVW_COLOR_COLOROBJ, ball);
        xvw_set_attribute(pseudo, XVW_COLOR_COLOROBJ, ball);

     /* display and run the program */
        xvf_run_form();
}
```

D.5. The Threshold Object

Figure 12: The threshold object allows you to perform pixel windowing and pixel thresholding on an image.

D.5.1. xvw_create_threshold() — *create a threshold object*

Synopsis

```
xvobject xvw_create_threshold(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the threshold object (for use in app-defaults files, etc)

Returns

The threshold object on success, NULL on failure

Description

Creates an threshold object which will display the colormap values of an object's colormap.

The threshold object is used by the user to mask out certain parts of the image so that they are only looking at desired range of pixels. The threshold objects contains an upper and lower scrollbar from which we can extract the pixel range to be threshold.

The threshold objects also contains the "pixel" that we are going to be setting the threshold values to, and whether we are to accept or reject that range. The "accept" value is a boolean which tells us if we are to threshold everything outside the lower & upper range or whether we threshold everything inside that range. (If accept is true then we threshold everything outside the range otherwise we threshold the pixels inside the range).

The threshold works by setting the pixels in the threshold range to the current threshold color (specified by pixel). And then storing these values in the xvdisplay colormap. But in order to make sure none of the other displays over-ride this threshold we set the pixel's active boolean array to zero. By setting the histogram to zero, none of the other displays will update those pixels with new values, until we restore the active states.

Note that thresholding should not be used with RGB images when the calling application is displayed on an 8-bit screen. This is because a mapping transformation is used in order to convert the RGB image into an 8-bit representation for visual display on an 8-bit screen.

Because of this transformation, thresholding an RGB image on an 8-bit screen will not make sense; any changes to the colormap made via thresholding would alter the 24-bit to 8-bit transformation (a 3-3-2 RGB map is used), rather than the RGB values themselves, producing an unexpected result.

D.5.2. Attributes of the Threshold Object

Summary of Threshold Attributes

Attribute	Description

XVW_THRESHOLD_CALLBACK	If desired, a callback may be installed on the threshold object that will be fired each time the user employs one of the scrollbars to change the lower or upper bound of the thresholding (or clipping) region. An *xvw_threshold_struct* will be passed in as the call_data; cast this parameter accordingly before using, as in:
	
	`xvw_threshold_struct *thresh_struct;`
	`thresh_struct = (xvw_threshold_struct *) call_data;`
	The *xvw_threshold_struct* is defined in $ENVISION/include/xvimage/Threshold.h as follows:
	
	`typedef struct`
	`{`
	`  xvobject intobj;`
	`  int  type;`
	`  int  value;`
	`} xvw_threshold_struct;`

XVW_THRESHOLD_CLIP_PIXELVAL	Used only when the XVW_THRESHOLD_POLICY attribute is set to KTHRESHOLD_POLICY_CLIP, this is the pixel value used in the clipped, or masked-out, regions of the image. All pixels inside the specified range will appear in their original pixel values. The opposite happens when XVW_THRESHOLD_INVERT is set to TRUE (ie, the pixel value is used for the pixels insie the range, while all pixels outside the range appear in their original pixel values).

Summary of Threshold Attributes	
Attribute	**Description**
XVW_THRESHOLD_INVERT	When the threshold object is set to "non-inverted" (ie, if XVW_THRESHOLD_THRES_INVERT is set to FALSE), the pixel thresholding, pixel clipping, and windowed thresholding operations will work normally. When the threshold object is set to "inverted" (ie, if XVW_THRESHOLD_THRES_INVERT is set to TRUE), the pixel thresholding, pixel clipping, and windowed thresholding operations will have the opposite effect from what they do normally. For pixel thresholding, the values outside the range (rather than inside) will be displayed in the pixel value specified; for pixel clipping, the values outside the range (rather than inside) will appear in their original values; for windowed thresholding, the values outside the range (rather than inside) will have their values normalized.
XVW_THRESHOLD_LOWERVAL	The lower value of the thresholding (or clipping) region.
XVW_THRESHOLD_PALETTE_OBJECT	This is the palette component of the threshold object. The palette component is where the actual pixels are displayed and reflect the thresholding process.
XVW_THRESHOLD_PIXELVAL	Used only when the XVW_THRESHOLD_POLICY attribute is set to KTHRESHOLD_POLICY_THRESH, this is the pixel value assigned to all the pixels within the range defined by the scroll bars of the threshold object. All pixels outside the specified range will appear black. The opposite happens when XVW_THRESHOLD_INVERT is set to TRUE (ie, the pixel value is used for the pixels outside the range, while all pixels inside the range are black).

Summary of Threshold Attributes	
Attribute	**Description**
XVW_THRESHOLD_POLICY	A threshold visual object provides a mechanism with which you may perform various types of thresholding on an image. When XVW_THRESHOLD_POLICY is set to KTHRESHOLD_POLICY_THRESH, the threshold visual object will perform pixel thresholding on the image. When XVW_THRESHOLD_POLICY is set to KTHRESHOLD_POLICY_CLIP, the threshold visual object will perform pixel clipping on the image. When XVW_THRESHOLD_POLICY is set to KTHRESHOLD_POLICY_WINDOWED, a windowed thresholding algorithm is used to increase the contrast of the image. **Pixel Clipping** Pixel clipping is used to mask out certain parts of the image sc is observed. Pixel values to be displayed; all values outside th predetermined pixel value. The user interactively specifies the retain their original values. The pixel value to be used for the v may be set using the XVW_THRESHOLD_CLIP_PIXELVAL attri **Pixel Thresholding** Pixel thresholding is used to set all values in the image to one specified range will be displayed using a non-zero pixel value, will be displayed in black.The user interactively specifies the r displayed in the non-zero pixel value. The non-zero pixel valu XVW_THRESHOLD_PIXELVAL attribute. **Window Contrast Enhancement** Windowed thresholding is used to increase the contrast of the while masking out the other pixels in the image. The user inte pixel values that will be displayed; all pixels outside the range ues inside the specified range will have their pixel values norm pixels in the image; thus, this operation can be thought of as a
XVW_THRESHOLD_RESET	This *action attribute* resets the lower and upper values of the threshold range to the lower an the effect of undoing any prior thresholding operations, and setting the data object back to it
XVW_THRESHOLD_SHOW_PALETTE	This attribute causes the palette of linear colormap values to be displayed above the lower/u palette is not to be shown.
XVW_THRESHOLD_UPPERVAL	The upper value of the thresholding (or clipping) region

Descriptions of Threshold Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_THRESHOLD_CALLBACK (N/A)	void (*callback_routine)(xvobject, kaddr, kaddr)	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_THRESHOLD_CLIP_PIXELVAL (thresholdClipPixelVal)	Pixel	white	any legal Pixel value
XVW_THRESHOLD_INVERT (thresholdInvert)	int	FALSE	TRUE/FALSE
XVW_THRESHOLD_LOWERVAL (N/A)	double	minval	minval - maxval
XVW_THRESHOLD_PALETTE_OBJECT (N/A)	xvobject	NULL	valid xvobject
XVW_THRESHOLD_PIXELVAL (thresholdPixelVal)	int	black	any legal Pixel value
XVW_THRESHOLD_POLICY (thresholdPolicy)	int	KTHRESHOLD_POLICY_CLIP	KTHRESHOLD_POLICY_THRESH KTHRESHOLD_POLICY_CLIP KTHRESHOLD_POLICY_WINDOWED
XVW_THRESHOLD_RESET (N/A)	int	N/A (action attribute)	TRUE
XVW_THRESHOLD_SHOW_PALETTE (thresholdShowPalette)	int	TRUE	TRUE/FALSE
XVW_THRESHOLD_UPPERVAL (N/A)	double	maxval	minval - maxval

D.5.3. Attributes of the Threshold Object

The inheritance tree of the threshold object is as follows:

manager -> graphics -> color -> threshold

Accordingly, the complete resource set for the threshold object includes:

1. The threshold object attribute, given above
2. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
3. The general object attributes, given in Chapter 2, "The xvwidgets Library", Section B, "General Attributes of GUI and Visual Objects".

D.5.4. Example using the Threshold Visual Object

An example using the threshold objec may be found in `$ENVISION/examples/color/threshold/1.threshold_display`.

```
/*
 *  This example displays the threshold object, and
 *  allows you to do pixel windowing and pixel thresholding
 *  on the mandril image.
 */
#include <envision.h>

void  quit_program PROTO((xvobject, kaddr, XEvent *, int *));

int main(
   int  argc,
   char *argv[])
{
        xvobject parent, image, threshold;
     kobject data_object;
     char *filename = "image:mandril";

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

        /* initialize xvwidgets lib */
        if (!xvw_initialize(XVW_MENUS_XVFORMS))
        {
           kerror(NULL, "main", "unable to open display");
           kexit(KEXIT_FAILURE);
        }

     /* open data object to be displayed in image & threshold objects */
     data_object = kpds_open_input_object(filename);

     /* create manager to be the parent object */
     parent = xvw_create_manager(NULL, "parent");

     /* create image object */
     image = xvw_create_image(parent, "image");
     xvw_set_attribute(image, XVW_IMAGE_IMAGEOBJ, data_object);

        /* create threshold object */
        threshold = xvw_create_threshold(parent, "threshold");
        xvw_set_attributes(threshold,
                     XVW_BELOW,          image,
                XVW_TACK_EDGE,      KMANAGER_TACK_HORIZ,
                XVW_COLOR_COLOROBJ, data_object,
                         NULL);

     /* display menuform for threshold object */
     xvw_activate_menu(threshold);

        /* add the action handler to quit on KeyPress 'q' */
        xvw_add_event(image, ButtonPressMask, quit_program, parent);

     /* display and run */
        xvf_run_form();
}
```

```
/*
 *  action handler to quit program on "q"
 */
void  quit_program(
   xvobject object,
   kaddr    client_data,
   XEvent   *event,
   int      *dispatch)
{
     xvobject parent = (xvobject) client_data;

        xvw_destroy(xvw_toplevel(parent));
}
```

This page left intentionally blank

Table of Contents

Chapter 6

Xvplot

Chapter 6 - Xvplot

A. Overview of Visual Objects Related To Plotting

A large variety of both 2D and 3D plot types are supported by the *plot2D* and *plot3D* visual objects. The *2D axis* object automatically creates and maintains the axes needed with 2D plots. Unfortunately, a *3D axis* object is not available as of yet, but is planned for the future. *Area* objects provide coordinated views in which interactive manipulation of plots, axes and annotations may be easily maintained. The visual objects related to plotting include: the 2D axis object, the 2D plot object, the 3D plot object, and the area object.

Available Functions

- *xvw_create_area()* - create a graphics area object
- *xvw_create_axis2d()* - create a 2D axis object
- *xvw_create_indicator()* - creates an indicator object
- *xvw_create_plot2d()* - create a 2D plot object
- *xvw_create_plot3d()* - create a 3D plot object

B. Issues Related to Plotting

Data to be plotted by a 2D or 3D plot object may be provided in one of two ways:

1. As an array of Coords, where the Coord structure is defined as:

```
typedef struct {
  Real  x, y, z;  /* X, Y, and Z coordinates of plot points */
  Index d;        /* index into colormap, determining plot point color */
} Coord
```

2. As a data object, as defined by the Polymorphic Data Model (see *Program Services Manual Vol II, Chapters 1 and 2*).

Whether or not the plot data originally comes from an array of Coords or from a data object, it is translated internally into a Coord array in preparation for plotting. When the plot data originates as an array of Coords, there is little "data interpretation" to be done. It is when the plot data originates as a data object that complexity arises; the interpretation of that data will depending on the contents of the data object.

In addition to the issue of data interpretation, there are a number of rules governing how a plot is colored that must be understood in order to obtain the best results from the 2D and 3D plot objects. Thus, data interpretation and plot color are the subjects to which the following sections are devoted.

B.1. Interpretation of the Data Object

Interpretation of the data contained in a data object by a 2D or 3D plot object depends on the contents of the data object. [1] The value segment, location segment, mask segment, and map segment may all be used to provide input to the 2D or 3D plot object; the time segment is the only data segment that is not interpreted. The value segment and the location segment of a data object are used to specify the data points that are plotted. If a map segment is present, it can be used to specify the colormap of the plot. A mask segment may also be used to indicate which points in the data are valid and should be plotted, and which points in the data are invalid, and should be omitted from the plot.

B.1.1. Value Segment Interpretation

Implicit Vs. Explicit Values

When the data provided for plot interpretation does not contain values to specify each coordinate of points in a plot (as is the case when only the value segment is present, and no location segment exists), some plot-point coordinates must be *implicit*. As opposed to an explicit value which is actually present in the data, an*implicit* value is implied by its location in the value segment. Implicit values are always incremental, beginning at 0 and increasing to the size of the dimension being plotted. For example, 2D plot data which originates as a line of value data will obtain the Y values explicitly from the value segment; however, the X values will be implied as 0, 1, 2, etc, up to the size of the line.

Plotting Width, Height, Depth, Time, or Elements

All five dimensions of the value segment defined by the Polymorphic Data Model are supported by the 2D and 3D plot objects. Accordingly, data to be plotted may be specified down width, height, depth, time, or elements. The 2D plot object plots a single line of data. By default, this line will be interpreted as the first row of data values, oriented down width. However, the `XVW_PLOT2D_X_ORIENTATION` attribute may be set to `KWIDTH`, `KHEIGHT`, `KDEPTH`, `KTIME`, or `KELEMENTS` in order to specify the orientation of the line that is extracted from the data object and used to create the 2D plot. The orientation of the line being plotted by the 2D plot object determines the meaning of the implicit X values of the plot. For instance, if a 2D plot is oriented down width, the implicit X values of the 2D plot will represent width; if it is oriented down height, the implicit X values of the plot will represent height. The explicit values actually obtained from the value segment are always used as the Y coordinates

The 3D plot object plots a region of data. By default, this region is interpreted as the first (width x height) plane of the value segment. The `XVW_PLOT2D_X_ORIENTATION` and `XVW_PLOT2D_Y_ORIENTATION` attributes are used to specify the meaning of the implicit X and Y values of the 3D plot. As in the 2D case, they may be set to `KWIDTH`, `KHEIGHT`, `KDEPTH`, `KTIME`, or `KELEMENTS` in order to cause the 3D plot object to plot width against height, or height against elements, or any other combination of the five value data dimensions. The explicit values

[1] For detailed descriptions of the value, location, map, and mask segments of a data object, see Chapter 2 of the *Program Services Manual Vol II, Polymorphic Data Services*; this discussion assumes that the reader is familiar with the Polymorphic Data Model used by VisiQuest 2001.

obtained from the value segment are always used as the Z coordinates.

Plot Size

When data is obtained from the value segment for plotting, the entire size of the value segment with respect to the dimension being plotted is used. For example, suppose a particular data object contains a value segment of width 50 and height 75. A 2D plot of the data oriented down width will always contain 50 points, one for each of the values in a row of the value data [2] ; if the plot is changed to be oriented down height, it will then have 75 points, one for each of the values in a column of the value data. The 3D case works similarly. For example, a 3D plot of the same data set oriented down width and height will have 70 rows of 50 points each, the entire plot containing 3500 points.

Offsets into Width, Height, Depth, Time, and Elements

Taking into account the rules for plot size detailed above, it is not surprising that offsets into the data are supported *provided that the offset does not refer to the dimension being plotted*. Again assume the case of a data object of width 50 and height 75. A 2D plot that is oriented down width may have a height offset value of anything from 0 to 74; varying the height offset allows any row of the value segment to be plotted. Attempts to change the width offset, however, will result in an error message, as the offset into the dimension being plotted must always be 0 in order to obtain the entire extent of data. For both the 2D and 3D plot objects, there is one attribute corresponding to each of the dimensions of the value segment that is used to set the offset in that dimension. For example, `XVW_PLOT2D_HEIGHT_OFFSET` is used to offset the height on a 2D plot, and `XVW_PLOT3D_DEPTH_OFFSET` is used to offset the depth on a 3D plot. Remember that it does not make sense to set an offset in a particular dimension of the value segment to anything greater than the size of that dimension. For example, it does not make sense to set a depth offset of 21 when the data object being used as input only has a depth of 20, or to set a time offset of 3 when the time size of the value segment is only 1. Errors will occur when such attempts are made.

Color Interpretation

The foreground color can be used to specify the color of a plot in its entirety. When this is the case, there is no "color interpretation" as such; the foreground color is specified, and the plot appears in that color (see Chapter 2, Section B.3).

Alternatively, the color of each point may be individually specified. The value segment of the data object may be used to specify plot colors (see Chapter 5, Section G.3.2), the values used for color are the same as those used for the last coordinates in the coordinate pair or coordinate triplet, except that they are first cast to type Index (unsigned long) and normalized between 0 and 255. In other words, for a 2D plot, the same values are used for *d* as are used for the Y coordinates; for a 3D plot, the same values are used for *d* as are used for the Z coordinates.

Summary Table

The following table summarizes the data interpretation of the value segment by the 2D and 3D plot objects.

[2] The exception to this is when a mask segment is used, a subject which will be discussed later

Summary of Plot Data Interpretation With Data Object Containing Value Data	
Plot Dimension	Interpretation
2D Plot	x : implicit and incremental, from 0 to w, h, d, t, or e y : explicit value data, obtained down w, h, d, t, or e d : explicit value data, same as y but normalized from (0 - 255)
3D Plot	x : implicit and incremental, from 0 to w, h, d, t, or e y : implicit and incremental, from 0 to w, h, d, t, or e z : explicit value data, obtained from (w x h), (h x d), (w x d), etc d : explicit value data, same as z but normalized from (0 - 255)

B.1.2. Location Segment Interpretation

The data that is plotted by the 2D and 3D plot objects may be explicitly stored in the location segment of a data object. When a location segment is present in a data object, the 2D and 3D plot objects will use the location segment as the coordinates to be plotted, whether or not a value segment also exists. In the presence of a location segment, the value segment will only be used for color interpretation, and then only if ShadeType is specified as Imagery.

The way that the location segment is interpreted depends on its dimensionality, or the size of each location vector. The 2D plot object will interpret location dimensions of 1 to 3, while the 3D plot object will interpret location dimensions of 2 to 4.

Implicit Vs. Explicit Values

In general, the main motivation for specification of plot data within the location segment of a data object is to *avoid* the use of implicit values, and to specify all coordinate values explicitly. However, in the two cases when the location segment dimensionality is not large enough to accomodate explicit storage of all values (see "Interpretation of Location Dimensions," below) it is sometimes necessary to use implicit values in order to compensate. When implicit values are used with location segment, they follow the same rules as implicit values in the value segment.

Plotting Width, Height, or Depth

All three dimensions of the location segment defined by the Polymorphic Data Model are supported by the 2D and 3D plot objects. Accordingly, data to be plotted may be specified down width, height, or depth. By default, the 2D plot object extracts a line down width, while the 3D plot object extracts a plane of (width x height). As when data from the value segment is being plotted, the `XVW_PLOT2D_X_ORIENTATION` is used by the 2D plot object, and both `XVW_PLOT2D_X_ORIENTATION` and `XVW_PLOT2D_Y_ORIENTATION` is used by the 3D plot object to specify the orientation of the plot data to be interpreted. When the location segment is used to store the plot data, however, `KTIME` and `KELEMENTS` are not valid settings for the orientation attributes, since time and elements are not dimensions of the location segment.

Plot Size

The rules for determining plot size from data stored in the location segment are identical to those used when the data is stored in the value segment.

That is, the size of the plot data is determined by the orientation of the plot data and the size of the corresponding dimension of the location segment.

Offsets into Width, Height, and Depth

The same rules apply with respect to offsets into width, height, and depth of the location segment as they do with respect to offsets into the value segment. In keeping with the definition of the location segment, however, the time and elements offsets do not apply.

Interpretation of Location Dimensions

Interpretation of location segment changes depending on whether or not a value segment is also present. The biggest difference in the interpretation is with respect to color. Specification of color may be done either with the value segment or in the location segment. The `XVW_PLOT2D_COLOR_???` attribute, which may have a value of `KPLOT2D_???` or `KPLOT2D_???` indicates whether the value segment or the location segment is to be used to specify the color of plot points. Of course, only when the data object has both a a value segment and a location segment does the `XVW_PLOT2D_COLOR_???` really take effect. When a data object has no location segment, the color specification must come from the value segment; when it has no value segment, the color specification must come from the location segment. However, when both are present, and ShadeType is set to Imagery, the color values will be obtained from the value segment; when ShadeType is set Elevation, the color values will be obtained from the location segment.

The other difference between interpretation of a data object that has only a location segment and a data object that has both a location segment and a value segment is when the location dimension is smaller than the dimension of the plot. The 2D plot object will interpret location dimensions of 1, 2, or 3. When the location dimension is 2 or 3, there is enough information to explicitly store values for both X and Y; however, when the location dimension is only 1, another interpretation must be applied. When the location dimension is larger than 3, the interpretation is done as if the dimension was only 3; the additional values are ignored.

Similarly with the 3D case, the 3D plot object will interpret location dimensions of 2, 3, or 4. When the location dimension is 3 or 4, there is enough information to explicitly store values for X, Y, and Z; however, when the location dimension is only 2, a more flexible interpretation must be implied in order to compensate for the missing value. When the location dimension is larger than 4, the interpretation is done as if the dimension was only 4; the additional values are ignored.

Summary Tables

The following tables provide a summary of the rules used by the 2D and 3D plot object to interpret location segment. Note that when a table entry reads "value," this is a shorthand for ’Explicit value obtained from value segment’ Please see the "Value Data Interpretation" Section for details relevant to 2D or 3D plotting.

Summary of 2D Plot Data Interpretation With Data Object Containing Location Data		
Data Object Contents	Value Segment Present And (Shade Type) set to (Imagery)	Value Segment Absent Or (Shade Type) set to (Elevation)
Location dimension = 1 {location x values}	x : location x y : value d : value	x : implicit y : location x d : location x
Location dimension = 2 {location (x,y) values}	x : location x y : location y d : value	x : location x y : location y d : location y
Location dimension = 3 {location (x,y,d) values}	x : location x y : location y d : value	x : location x y : location y d : location d

Summary of 3D Plot Data Interpretation With Data Object Containing Location Data		
Data Object Contents	Value Segment Present And (Shade Type) set to (Imagery)	Value Segment Absent Or (Shade Type) set to (Elevation)
Location dimension = 1 {location x values}	invalid	invalid
Location dimension = 2 {location (x,y) values}	x : location x y : location y y : value d : value	x : implicit y : location x z : location y d : location y
Location dimension = 3 {location (x,y,z) values}	x : location x y : location y z : location z d : value	x : location x y : location y z : location z d : location z
Location dimension = 4 {location (x,y,z,d) values}	x : location x y : location y z : location z d : value	x : location x y : location y z : location z d : location d

B.2. Plot Color

Plots can be colored in one of two ways: the color of the entire plot can be specified with the foreground color, or the color of each point in the plot can be specified individually. In either case, the method for color specification differs depending on whether the plot data is specified using a Coord array of points or using a data object. Since 2D and 3D plots follow the same rules for color interpretation, the 2D case is used for this explanation.

Color Origination When Plot Data Is Specified Using Data Object

When the plot data is specified using a data object, the XVW_PLOT2D_COLOR_ORIGINATION attribute is used to specify whether the color of the plot is to be dictated by values stored in the data or by the foreground color. By default, this attribute have a value of KPLOT2D_COLOR_FROM_FOREGROUND; thus, by default, plots that are specified using a data object will be displayed in the foreground color. The foreground color can be specified either in an app-defaults file (see Appendix A) or using the XVW_FOREGROUND_COLOR attribute.

Alternatively, the color specification for each point in the plot may be provided in a specified colormap. If this is to be the case, the XVW_PLOT2D_COLOR_ORIGINATION must first be set to KPLOT2D_COLOR_FROM_DATA.

When the color specification for the plot is included as part of the plot data, there may be one of three cases. In the first case, the data object being plotted contains its own colormap. When this is the case, that colormap will dictate the color of each point in the plot. In the second case, the data object being plotted does not contain its own colormap; for lack of more specific information, the plot object will use the Rainbow colormap to color the plot. In the third case, the data object being plotted may or may not contain its own colormap, but this information is ignored because it is over-ridden by an explicit set of the XVW_COLOR_COLOROBJ attribute by the application. If this resource is explicitly set on the plot object, the colormap contained by the data object referenced will always over-ride any other color interpretation possibilities.

The following table summarizes the rules for color interpretation when a data object is used to provide the plot data. In this table, the category "Use Foreground" implies that XVW_PLOT2D_COLOR_ORIGINATION is set to the default KPLOT2D_COLOR_FROM_FOREGROUND; the category "Use Data" implies that XVW_PLOT2D_COLOR_ORIGINATION has been set to KPLOT2D_COLOR_FROM_DATA. In the "Configuration" entries, "plot object" means the data object supplying the data to be plotted, specified using KPLOT2D_PLOTOBJ, and "color object" means the data object supplying an alternate colormap, specified using XVW_COLOR_COLOROBJ.

Summary of Plot Color Origination When Plot Data Comes From a Data Object

Configuration	Use FG	Use Data
Plot object does contain a colormap; color object IS NOT set	Color dictated by foreground color	Color dictated by colormap in plot object
Plot object does not contain a colormap; color object IS NOT set	Color dictated by foreground color	Rainbow autocolor procedure is used
Plot object does contain a colormap; color object IS set	Color dictated by foreground color	Color dictated by colormap in color object
Plot object does not contain a colormap; color object IS set	Color dictated by foreground color	Color dictated by colormap in color object

Color Origination When Plot Data Is Specified Using Coord Array

When plot data is specified using a Coord array, the XVW_PLOT2D_COLOR_ORIGINATION attribute is not relevant. Instead, the Index *d* field of each Coord element is *always* used to dictate plot color. If the foreground color is to be used to color the plot, *the d value of each Coord element MUST be explicitly set to* KGRAPHICS_UNINITIALIZED. *When the d value is set to* KGRAPHICS_UNINITIALIZED, *the plot object will know to display that point of the plot in the foreground color. Naturally, if ALL Coord elements in the array have their d fields set in this way, the entire plot will appear in the foreground color.*

Alternatively, the *d* fields of the Coords in the array may be set to some value other than KGRAPHICS_UNINITIALIZED. When this is the case, the plot point in question will appear in the color implied by the value of the *d* field. Of course, in order to "imply" a color, the *d* fields of the Coords must contain indices into a colormap. Accordingly, a data object containing the colormap that will be indexed by the *d* fields MUST be specified using XVW_COLOR_COLOROBJ, or the colors in which the plot is displayed will be undefined. The following table summarizes the rules for color interpretation when a Coord array is used to provide the plot data. In this table, "color object" means the data object supplying the colormap, specified using XVW_COLOR_COLOROBJ.

Summary of Plot Color Origination When Plot Data Comes From a Coord Array

Configuration	Color Interpretation
Coord element "d" values set to KGRAPHICS_UNINITIALIZED	Plot point specified by that Coord element will appear in the foreground color
Coord element "d" values set to something other than KGRAPHICS_UNINITIALIZED; color object set to data object containing colormap	Coord element "d" values interpreted as indices into the colormap contained in the color object; plot points will appear in the color specified by the colormap.
Coord element "d" values set to something other than KGRAPHICS_UNINITIALIZED; color object NOT set to data object containing colormap	Coord element "d" values interpreted as indices into nonexistent colormap; results undefined

C. The Area Object

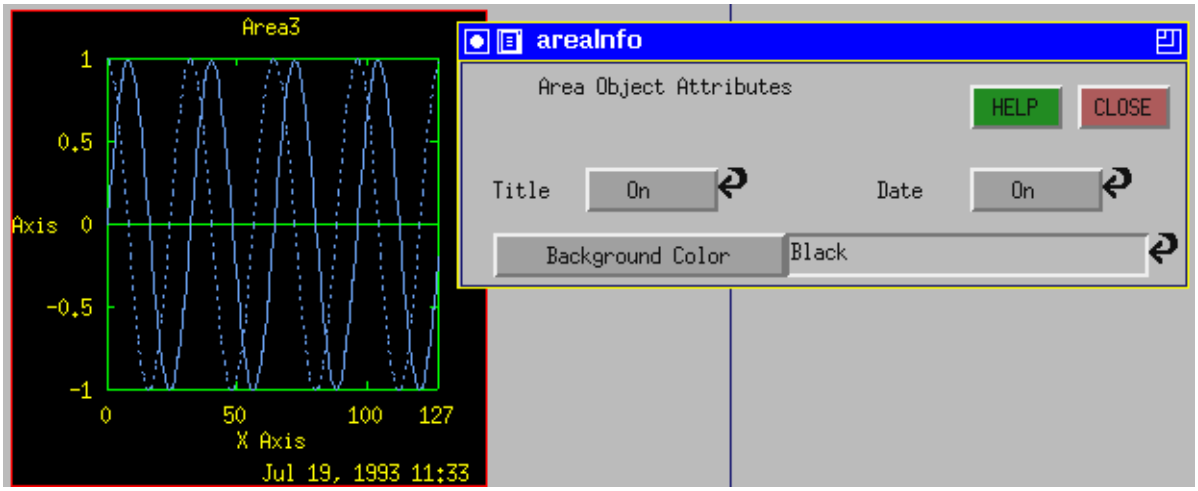


Figure 1: The area object is used to maintain a coordinated viewing system among the other objects to which it is a parent, most commonly plot objects and axis objects. Here, an area object is used to manage two 2D plot objects, an axis object, and a date object.

C.1. xvw_create_area() — *create a graphics area object*

Synopsis

```
xvobject xvw_create_area(
    xvobject parent,
    char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the area object (for use in app-defaults files, etc)

Returns

The area object on success, NULL on failure

Description

An area object provides an area which contains a coordinated view of other visual objects; it is primarily designed to provide a backplane for 2D plot objects, 3D plot objects, 2D axis objects, and annotations. Any visual objects that are children of the area object will have the same world view as the controlling visual object, where the controlling visual object may be specified by the application using the XVW_GRAPHICS_ATTACH attribute (if an axis is used, it is usually specified as the controlling visual object).

C.2. Attributes of the Area Object

Summary of Area Attributes	
Attribute	**Description**
XVW_AREA_DATE	If XVW_AREA_DISPLAY_DATE is set to TRUE, this *read-only* attribute attribute may be used to obtain the date visual object which is used to display the date.
XVW_AREA_DISPLAY_DATE	This attribute controls whether or not the date is displayed at the bottom of the area object. Set to TRUE to display date, FALSE to suppress date.
XVW_AREA_DISPLAY_TITLE	This attribute controls whether or not a title is displayed at the top of the area object. Set to TRUE to display title, FALSE to suppress title.
XVW_AREA_TITLE	If XVW_AREA_DISPLAY_TITLE is set to TRUE, this *read-only* attribute attribute may be used to obtain the string visual object which is used to display the title.
XVW_AREA_TITLE_STRING	If XVW_AREA_DISPLAY_TITLE is set to TRUE, this attribute determines the string that is used as the title of the area object.

Descriptions of Area Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_AREA_DATE (N/A)	xvobject	NULL	date xvobject (read only)
XVW_AREA_DISPLAY_DATE (areaDisplayDate)	int	FALSE	TRUE/FALSE
XVW_AREA_DISPLAY_TITLE (areaDisplayTitle)	int	TRUE	TRUE/FALSE
XVW_AREA_TITLE (N/A)	xvobject	NULL	string xvobject (read only)
XVW_AREA_TITLE_STRING (areaTitle.stringString)	char *	"Area Object"	any valid string

C.3. Resource Set of the Area Object

The inheritance tree of the animation object is as follows:

manager -> graphics -> area

Accordingly, the complete resource set for the area object includes:

1. The area object attribute, given above

2. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

C.4. Example using the Area Visual Object

Several example programs using the area visual object can be found in `$ENVISION/examples/plot/`. One of these is as follows.

```
#include <envision.h>

Coord  data[4000];

void  recompute_points PROTO((xvobject, kaddr, XEvent *, int *));

/*
 * This example demonstrates how an area object can be used to provide
 * a coordinated world coordinate system for a plot and a set of axes.
 * It creates an area object, with a 2D plot and a set of 2D axes within
 * it;  it attaches the plot and the axes to the area object, so they
 * have a coordinated world coordinate view, and keep that coordinated
 * view even when the range of the plot changes.
 *
 * Click on the image to invoke the event handler which re-computes the
 * sin curve plotted to a new range;  the plot and the axis system will
 * update together.
 */
#define PTNUM 50
void main(
   int  argc,
   char *argv[])
{
     xvobject parent, area, axis, plot;
     int    i, degrees;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

     /* initialize xvwidgets lib */
        if (!xvw_initialize(XVW_MENUS_XVFORMS))
        {
        kerror(NULL, "main", "Cannot open display");
        kexit(KEXIT_FAILURE);
        }

     /*
      *  Generate initial sin curve to be plotted
      */
     for (i = 0, degrees = 9; i < PTNUM; i++, degrees+=20)
     {
        data[i].x = i;
        data[i].y = sin((double) kdegrees_radians(degrees));
```

```
        data[i].d = KGRAPHICS_UNINITIALIZED;
    }

    /*
     * create a parent for the area
     */
    parent = xvw_create_manager(NULL, "parent");
    xvw_set_attributes(parent,
                XVW_WIDTH,  400,
                XVW_HEIGHT, 400,
                NULL);
       /*
        * create a coordinated axis and plot object in an area object
     */
    area = xvw_create_area(parent, "area");
    xvw_set_attributes(area,
                          XVW_AREA_DISPLAY_DATE,  FALSE,
                          XVW_AREA_TITLE_STRING, "2D Plot",
                   XVW_TACK_EDGE,          KMANAGER_TACK_ALL,
                          XVW_BACKGROUND_COLOR,   "black",
                          NULL );

    /*
     *  create the axis system
     */
    axis = xvw_create_axis2d(area, "axis");
    xvw_set_attributes(axis,
                XVW_GRAPHICS_ATTACH, axis,
                   XVW_GRAPHICS_VIEWPORT_MIN_X, 0.2,
                          XVW_GRAPHICS_VIEWPORT_MIN_Y, 0.2,
                          XVW_GRAPHICS_VIEWPORT_MAX_X, 0.9,
                          XVW_GRAPHICS_VIEWPORT_MAX_Y, 0.9,
                NULL);


    /*
     * The viewport max and mins do not have to be set for the plot
     * object, since it's attached to the axis object. They will have the
     * same values as the axis object, since the parent of the axis
        * object is an area object.  This is also true for the world
        * coordinates; however it is the plot that is dictating these values
        * (i.e. the maximum and minimum of the data being plotted) and the
        * axis that is inheriting them.
     */
    plot = xvw_create_plot2d(area, "plot coordinated");
    xvw_set_attributes(plot,
                XVW_GRAPHICS_ATTACH,   axis,
                XVW_PLOT2D_POINTS,      data,
                   XVW_PLOT2D_PLOTSIZE,       PTNUM,
                          XVW_FOREGROUND_COLOR,  "Magenta",
                   NULL);

       xvw_add_event(area, ButtonPressMask, recompute_points, plot);

    xvf_run_form();
}

void  recompute_points(
    xvobject object,
```

```
    kaddr    client_data,
    XEvent   *event,
    int      *dispatch)
{
        int        i, degrees;
        static int factor = 2;
     xvobject   plot_object = (xvobject) client_data;

        /*
         *  Generate a new sin curve to be plotted
         */
        for (i = 0, degrees = 9; i < (PTNUM * factor); i++, degrees+=20)
        {
            data[i].x = i;
            data[i].y = sin((double) kdegrees_radians(degrees));
         data[i].d = KGRAPHICS_UNINITIALIZED;
        }
        xvw_set_attributes(plot_object,
                    XVW_PLOT2D_POINTS,          data,
                    XVW_PLOT2D_PLOTSIZE,        (PTNUM * factor),
                    NULL);
        factor++;
        if ( factor > 5 ) factor = 1;
}
```

D. The 2D Plot Object

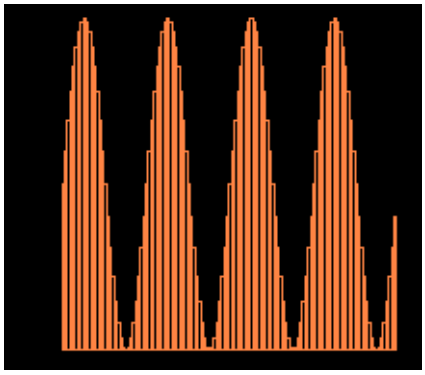


Figure 2: The 2D plot object supports a number of different plot types, including the histogram shown here. Its internal menuforms (not shown) allow the user to change the plot type, line type, marker type, and plot color.

D.1. xvw_create_plot2d() — *create a 2D plot object*

Synopsis

```
xvobject xvw_create_plot2d(
```

```
    xvobject parent,
    char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the 2D plot object (for use in app-defaults files, etc)

Returns

Returns the created 2D plot xvobject, or NULL upon failure

Description

A 2D plot object is used to display a 2D plot.

A variety of plot types are supported, including line plots, discrete plots, bar graphs, polymarker plots, linemarker plots, and colormarker plots. The 2D plot object is generally used in conjunction with a 2D axis object, where both are created as children of an area object so that a coordinated world coordinate view is automatically supported.

D.2. Attributes of the 2D Plot Object

Summary of Plot2D Attributes

Attribute	Description
PLOT2D_COLOR_PTS_INTERACTIVELY	This *action attribute* allows the user to rubberband about desired points in the plot and colors them automatically, in the color specified by XVW_PLOT2D_HIGHLIGHTCOLOR. As an action attribute, this attribute can only be used with *xvw_set_attribute(s)()*, not with *xvw_get_attribute(s)()*. When this attribute is set, the user is expected to move the pointer into the 2D plot object; the cursor will change to the hand, indicating that the user should rubber-band about the points that they wish to be colored differently in the plot. The ROI shape that is used for rubberbanding about the points is specified using the XVW_PLOT2D_ROI_SHAPE attribute. After the user has completed the rubber-banding procedure, the points inside the ROI specified are automatically colored in the highlight color. The points in the plot that are ouside the rubberbanded area will remain in the original color.

Summary of Plot2D Attributes	
Attribute	**Description**
PLOT2D_DELETE_PTS_INTERACTIVELY	This *action attribute* allows the user to rubberband about desired points in the plot and deletes them automatically. As an action attribute, this attribute can only be used with *xvw_set_attribute(s)()*, not with *xvw_get_attribute(s)()*. When this attribute is set, the user is expected to move the pointer into the 2D plot object; the cursor will change to the hand, indicating that the user should rubber-band about the points that they wish to be deleted from the plot. The ROI shape that is used for rubberbanding about the points is specified using the XVW_PLOT2D_ROI_SHAPE attribute. After the user has completed the rubber-banding procedure, the points inside the ROI specified are automatically deleted from the plot data. Thus, setting this action attribute has the side effect of modifying the data being used by the 2D plot object; if the data has been specified using the XVW_PLOT2D_POINTS attribute, the pointer to the Coord array should be re-obtained with *xvw_get_attribute(s)()* before it is used again by the application.
XVW_PLOT2D_COLOR_ORIGINATION	This attribute is only used when the plot data is provided via a kobject specified using the XVW_PLOT2D_PLOTOBJ or XVW_PLOT2D_PLOTFILE attributes. It dictates whether the color of the plot is to be specified by the plot foreground color or by the plot data itself. If the entire plot is to appear in a particular color, set this attribute to KPLOT2D_COLOR_FROM_FOREGROUND; then, specify the desired color using the XVW_FOREGROUND or XVW_FOREGROUND_COLOR attributes. If each point in the data set is to have its color specified individually, set this attribute to KPLOT2D_COLOR_FROM_DATA. The color of each point in the plot will be dictated by the values stored in the kobject; see ??? for details on how the color is interpreted from the data of the kobject.
XVW_PLOT2D_DATA_MAX_X	This *read-only* attribute returns the maximum of the X coordinates in the 2D plot data currently in the Coord array specified with XVW_PLOT2D_POINTS.
XVW_PLOT2D_DATA_MAX_Y	This *read-only* attribute returns the maximum of the Y coordinates in the 2D plot data currently in the Coord array specified with XVW_PLOT2D_POINTS.
XVW_PLOT2D_DATA_MIN_X	This *read-only* attribute returns the minimum of the X coordinates in the 2D plot data currently in the Coord array specified with XVW_PLOT2D_POINTS.
XVW_PLOT2D_DATA_MIN_Y	This *read-only* attribute returns the minimum of the Y coordinates in the 2D plot data currently in the Coord array specified with XVW_PLOT2D_POINTS.
XVW_PLOT2D_DEPTH_OFFSET	This attribute in conjunction with the XVW_PLOT2D_X_ORIENTATION attribute; when XVW_PLOT2D_X_ORIENTATION is *not* set to KDEPTH, it is used to specify the depth offset at which the line of plot data is to be extracted.

Summary of Plot2D Attributes	
Attribute	**Description**
XVW_PLOT2D_ELEMENTS_OFFSET	This attribute in conjunction with the XVW_PLOT2D_X_ORIENTATION attribute; when XVW_PLOT2D_X_ORIENTATION is *not* set to KELEMENTS, it is used to specify the elements offset at which the line of plot data is to be extracted.
XVW_PLOT2D_HEIGHT_OFFSET	This attribute in conjunction with the XVW_PLOT2D_X_ORIENTATION attribute; when XVW_PLOT2D_X_ORIENTATION is *not* set to KHEIGHT, it is used to specify the height offset at which the line of plot data is to be extracted.
XVW_PLOT2D_HIGHLIGHTCOLOR	When the PLOT2D_COLOR_PTS_INTERACTIVELY attribute is used to interactively change the color of a region in the plot within a particular region of interest, this attribute specifies the color in which that portion of the plot will be displayed. The color is specified by name.
XVW_PLOT2D_PLOTFILE	The file containing the 2D plot data to be displayed. Note that this attribute is mutually exclusive with XVW_PLOT2D_PLOTOBJ; use one or the other, not both.
XVW_PLOT2D_PLOTOBJ	The data object containing the 2D plot data to be displayed. Note that this attribute is mutually exclusive with XVW_PLOT2D_PLOTFILE; use one or the other, not both.
XVW_PLOT2D_PLOTSIZE	This is the number of data points contained in the Coord array specified by the attribute XVW_PLOT2D_POINTS. Note that you *must* use this attribute to specify the number of points prior to specifying the data points with XVW_PLOT2D_POINTS.
XVW_PLOT2D_PLOTTYPE	This attribute indicates how the plot is to be displayed. Plot types include line plots, discrete plots, bar graphs, polymarker plots (also known as scatter plots), and linemarker plots (a combination of the line plot and the scatter plot).

Summary of Plot2D Attributes	
Attribute	**Description**

XVW_PLOT2D_POINTS	This is the array of coordinates defining the 2D data points in the plot. It is an array of type Coord, where the Coord structure is defined as:
	
	`typedef struct`
	`{`
	`  Real x, y, z;`
	`  Index d;`
	`} Coord;`
	Note that the XVW_PLOT2D_PLOTSIZE attribute *must* be set to the number of points in the Coord array *prior* to setting the XVW_PLOT2D_POINTS attribute.
	The *z* value is ignored for all 2D plot types.
	If the color of a plot point is to be dictated by either the XVW_FOREGROUND or the XVW_FOREGROUND_COLOR attributes, set the *d* value to KGRAPHICS_UNINITIALIZED.
	
	Alternatively, you may set the *d* value to the pixel value which specifies the desired color of the plot point. See ??? for more details on specifying plot colors.

XVW_PLOT2D_ROI	This is a read-only action attribute; that is, it can only be used with *xvw_get_attribute(s)()*. It allows the user to interactively specify a desired region of interest in the 2D plot object. The plot points that are inside the specified region of interest will be returned in a coordinate array.
XVW_PLOT2D_ROI_POLICY	When the XVW_PLOT2D_ROI attribute is used to extract a region of interest, this attribute specifies whether the ROI is defined by the region inside the shape, by the region outside the shape, or by the outline of the shape itself.
XVW_PLOT2D_ROI_SHAPE	When the XVW_PLOT2D_ROI attribute is used to extract the plot points that fall within a particular region of interest, this attribute specifies the shape of the ROI that will be interactively drawn by the user. Supported ROI shapes include rectangle, polygon, circle, ellipse, line, and freehand.
XVW_PLOT2D_TIME_OFFSET	This attribute in conjunction with the XVW_PLOT2D_X_ORIENTATION attribute; when XVW_PLOT2D_X_ORIENTATION is *not* set to KTIME, it is used to specify the time offset at which the line of plot data is to be extracted.
XVW_PLOT2D_WIDTH_OFFSET	This attribute in conjunction with the XVW_PLOT2D_X_ORIENTATION attribute; when XVW_PLOT2D_X_ORIENTATION is *not* set to KWIDTH, it is used to specify the width offset at which the line of plot data is to be extracted.

Summary of Plot2D Attributes	
Attribute	**Description**
XVW_PLOT2D_X_ORIENTATION	This attribute is only used when the plot data is provided via a kobject specified using the XVW_PLOT2D_PLOTOBJ or XVW_PLOT2D_PLOTFILE attributes; furthermore, it only applies when the plot data is stored in the value segment of the data object. It dictates how the 2D plot data is to be extracted from the value segment of the data object. The 2D plot data is always extracted as a single line, but that line may be oriented down width, height, depth, time, or elements.

Descriptions of Plot2D Attributes

Attribute (Resource Name)	Type	Default	Legal Values
PLOT2D_COLOR_PTS_INTERACTIVELY (N/A)	int	N/A	TRUE
PLOT2D_DELETE_PTS_INTERACTIVELY (N/A)	int	N/A	TRUE
XVW_PLOT2D_COLOR_ORIGINATION (plot2DColorOrigination)	int	KPLOT2D_COLOR_FROM_FOREGROUND	KPLOT2D_COLOR_FROM_FOREGROUND or symbol index KPLOT2D_COLOR_FROM_DATA
XVW_PLOT2D_DATA_MAX_X (N/A)	double	1.0	the maximum X world coordinate value (read only)
XVW_PLOT2D_DATA_MAX_Y (N/A)	double	1.0	the maximum Y world coordinate value (read only)
XVW_PLOT2D_DATA_MIN_X (N/A)	double	0.0	the minimum X world coordinate value (read only)
XVW_PLOT2D_DATA_MIN_Y (N/A)	double	0.0	the minimum Y world coordinate value (read only)
XVW_PLOT2D_DEPTH_OFFSET (N/A)	int	0	0 < value < depth of data object value segment
XVW_PLOT2D_ELEMENTS_OFFSET (N/A)	int	0	value < number of elements in data object value segment
XVW_PLOT2D_HEIGHT_OFFSET (N/A)	int	0	0 < value < height of data object value segment
XVW_PLOT2D_HIGHLIGHTCOLOR (plot2DHighlightColor)	char *	"red"	any valid color name: see /usr/lib/X11/rgb.txt or Section 7.1.1 of *The XLib Programming Manual* by Adrian Nye
XVW_PLOT2D_PLOTFILE (N/A)	char *	NULL	any valid input file
XVW_PLOT2D_PLOTOBJ (N/A)	kobject	NULL	any valid data object

Descriptions of Plot2D Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_PLOT2D_PLOTSIZE (N/A)	int	0	values > 0
XVW_PLOT2D_PLOTTYPE (plot2DPlotType)	int	KPLOT2D_LINEPLOT KPLOT2D_BARGRAPH KPLOT2D_DISCRETE KPLOT2D_POLYMARKER KPLOT2D_LINEMARKER	
XVW_PLOT2D_POINTS (N/A)	Coord *	NULL	array of Coords defining data points for 2D plot
XVW_PLOT2D_ROI (N/A)	Coord *	NULL	Those points that fall inside the specified ROI
XVW_PLOT2D_ROI_POLICY (imageRoiPolicy)	int	KPLOT2D_ROI_INSIDE	KPLOT2D_ROI_INSIDE KPLOT2D_ROI_OUTLINE KPLOT2D_ROI_OUTSIDE
XVW_PLOT2D_ROI_SHAPE (plot2DRoiShape)	int	KPLOT2D_ROI_RECTANGLE	KPLOT2D_ROI_RECTANGLE KPLOT2D_ROI_POLYLINE KPLOT2D_ROI_CIRCLE KPLOT2D_ROI_ELLIPSE KPLOT2D_ROI_LINE KPLOT2D_ROI_FREEHAND
XVW_PLOT2D_TIME_OFFSET (N/A)	int	0	value < time size of data object value segment
XVW_PLOT2D_WIDTH_OFFSET (N/A)	int	0	0 < value < width of data object value segment
XVW_PLOT2D_X_ORIENTATION (N/A)	int	KWIDTH	KWIDTH KHEIGHT KDEPTH KTIME KELEMENTS

D.3. Complete Resource Set of the 2D Plot Object

The inheritance tree of the 2D plot object is as follows:

manager -> graphics -> color -> plot2D

The complete resource set for the 2D plot object includes:

1. The 2D plot attribute resource set, given above
2. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

D.4. Example Using the 2D Plot Visual Object

Examples of programs using the 2D Plot object can be found in `$ENVISION/examples/plot/`. One of these is as follows.

```
#include <envision.h>

/*
 * This simple program plots the data contained in
 * a VIFF file, in red on a black background
 */

void main(
   int  argc,
   char *argv[])
{
     xvobject plot;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

     /* initialize the xvwidgets lib */
        if (!xvw_initialize(XVW_MENUS_XVFORMS))
        {
        kerror(NULL, "main", "Cannot open display");
        kexit(KEXIT_FAILURE);
        }

        /*
         * create the plot object; set the file that contains the data
         * for the plot, and set the foreground color.
         */
     plot = xvw_create_plot2d(NULL, "plot");
     xvw_set_attributes(plot,
                  XVW_PLOT2D_PLOTFILE,  "image:featheye",
                  XVW_FOREGROUND_COLOR, "red",
                     NULL);

     /* want the background to be black */
     xvw_set_attribute(xvw_parent(plot), XVW_BACKGROUND_COLOR, "black");

     /* display and run */
     xvf_run_form();
}
```

E. The 3D Plot Object

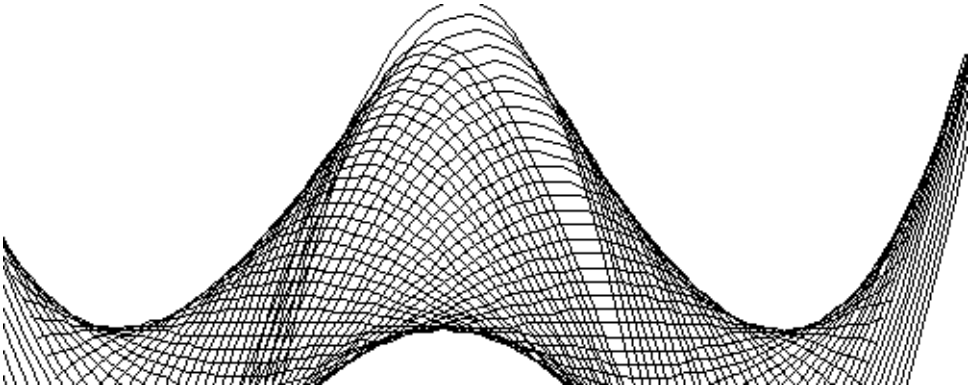


Figure 3: The 3D plot object supports a number of different plot types, including the line plot shown here. Its internal menuforms (not shown) allow the user to change a wide variety of 3D plot attributes.

E.1. xvw_create_plot3d() — *create a 3D plot object*

Synopsis

```
xvobject xvw_create_plot3d(
    xvobject parent,
    char   *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the 3D plot object (for use in app-defaults files, etc)

Returns

Returns the created 3D plot xvobject, or NULL upon failure

Description

A 3D plot object is used to display a 3D plot. A variety of plot types are supported, including line plots, mesh plots, scatter plots, 2D and 3D contour plots, impulse plots, horizon plots, wireframe plots, and shaded plots.

Ideally, the 3D plot object would be used in conjunction with a 3D axis object, where both were created as children of an area object so that a coordinated world coordinate view is automatically supported. Unfortunately, the 3D axis object has not yet been developed (sorry, folks).

E.2. Attributes of the 3D Plot Object

Summary of Plot3D Attributes	
Attribute	**Description**
XVW_PLOT3D_COLOR_ORIGINATION	This attribute is only used when the plot data is provided via a kobject specified using the XVW_PLOT3D_PLOTOBJ or XVW_PLOT3D_PLOTFILE attributes. It dictates whether the color of the plot is to be specified by the plot foreground color or by the plot data itself. If the entire plot is to appear in a particular color, set this attribute to KPLOT3D_COLOR_FROM_FOREGROUND; then, specify the desired color using the XVW_FOREGROUND or XVW_FOREGROUND_COLOR attributes. If each point in the data set is to have its color specified individually, set this attribute to KPLOT3D_COLOR_FROM_DATA. The color of each point in the plot will be dictated by the values stored in the kobject; see ??? for details on how the color is interpreted from the data of the kobject.
XVW_PLOT3D_CONTOUR_LEVELS	This attribute may be set to an array of Real values that specify the Z values of the contour levels to be drawn when XVW_PLOT3D_PLOTTYPE is set to KPLOT3D_CONTOUR_3D or KPLOT3D_CONTOUR_2D. When using this attribute, the XVW_PLOT3D_NUM_CONTOUR_LEVELS attribute must be used to specify the size of the Real array.
XVW_PLOT3D_DATA_MAX_X	This *read-only* attribute returns the maximum of the X coordinates in the 3D plot data currently in the Coord array specified with XVW_PLOT3D_POINTS.
XVW_PLOT3D_DATA_MAX_Y	This *read-only* attribute returns the maximum of the Y coordinates in the 3D plot data currently in the Coord array specified with XVW_PLOT3D_POINTS.
XVW_PLOT3D_DATA_MAX_Z	This *read-only* attribute returns the maximum of the Z coordinates in the 3D plot data currently in the Coord array specified with XVW_PLOT3D_POINTS.
XVW_PLOT3D_DATA_MIN_X	This *read-only* attribute returns the minimum of the X coordinates in the 3D plot data currently in the Coord array specified with XVW_PLOT3D_POINTS.
XVW_PLOT3D_DATA_MIN_Y	This *read-only* attribute returns the minimum of the Y coordinates in the 3D plot data currently in the Coord array specified with XVW_PLOT3D_POINTS.
XVW_PLOT3D_DATA_MIN_Z	This *read-only* attribute returns the minimum of the Z coordinates in the 3D plot data currently in the Coord array specified with XVW_PLOT3D_POINTS.
XVW_PLOT3D_DEPTH_OFFSET	This attribute is used in conjunction with the XVW_PLOT3D_X_ORIENTATION and XVW_PLOT3D_Y_ORIENTATION attributes. When *neither* of these attributes is set to KDEPTH, it is used to specify the depth offset at which the X and Y coordinates for the plot data are to be extracted.

Summary of Plot3D Attributes	
Attribute	**Description**
XVW_PLOT3D_ELEMENTS_OFFSET	This attribute is used in conjunction with the XVW_PLOT3D_X_ORIENTATION and XVW_PLOT3D_Y_ORIENTATION attributes. When *neither* of these attributes is set to KELEMENTS, it is used to specify the elements offset at which the X and Y coordinates for the plot data are to be extracted.
XVW_PLOT3D_HEIGHT_OFFSET	This attribute is used in conjunction with the XVW_PLOT3D_X_ORIENTATION and XVW_PLOT3D_Y_ORIENTATION attributes. When *neither* of these attributes is set to KHEIGHT, it is used to specify the height offset at which the X and Y coordinates for the plot data are to be extracted.
XVW_PLOT3D_NUM_CONTOUR_LEVELS	The number of contour levels to be drawn when XVW_PLOT3D_PLOTTYPE is set to KPLOT3D_CONTOUR_3D or KPLOT3D_CONTOUR_2D. If set to the default (-1) value, there will be one contour level drawn for each Z value of the plot data, where the Z values of the contour levels are calculated automatically. If the number of unique Z values in the plot data exceeds the number of colors that can be displayed, there will be one contour level drawn for each color that can be displayed. When the KPLOT3D_CONTOUR_LEVELS is set to an array of Real values that specify the Z values of the contour levels, XVW_PLOT3D_NUM_CONTOUR_LEVELS is used to indicate the size of the array; If KPLOT3D_CONTOUR_LEVELS is set to NULL (the default value) and XVW_PLOT3D_NUM_CONTOUR_LEVELS is set to a value greater than 0, the Z values of the contour levels will be automatically calculated.
XVW_PLOT3D_PLOTFILE	The file containing the 3D plot data to be displayed. Note that this attribute is mutually exclusive with XVW_PLOT3D_PLOTOBJ; use one or the other, not both.
XVW_PLOT3D_PLOTOBJ	The data object containing the 3D plot data to be displayed. Note that this attribute is mutually exclusive with XVW_PLOT3D_PLOTFILE; use one or the other, not both.
XVW_PLOT3D_PLOTSIZE	This is the number of data points contained in the Coord array specified by the attribute XVW_PLOT3D_POINTS. Note that you *must* use this attribute to specify the number of points prior to specifying the data points with XVW_PLOT3D_POINTS.
XVW_PLOT3D_PLOTTYPE	This attribute indicates how the plot is to be displayed. Choices include: KPLOT3D_LINEPLOT KPLOT3D_MESH KPLOT3D_SCATTER KPLOT3D_CONTOUR_3D KPLOT3D_CONTOUR_2D KPLOT3D_IMPULSE KPLOT3D_PHONG_SHADING KPLOT3D_GHOURAUD_SHADING KPLOT3D_CONSTANT_SHADING KPLOT3D_HORIZON KPLOT3D_WIREFRAME

Summary of Plot3D Attributes	
Attribute	**Description**
XVW_PLOT3D_PLOTWIDTH	This attribute dictates the number of data points in each *row* to be plotted. Thus, if there are 100 data points specified by XVW_PLOT3D_PLOTSIZE and XVW_PLOT3D_PLOTWIDTH is set to 2, this implies that the 3D plot has 50 continuous lines, each with 2 data points. Alternatively, if XVW_PLOT3D_PLOTWIDTH was set to 25, this would imply that the same 100 data triplets were to be plotted as 4 continuous lines, each with 25 data points.
XVW_PLOT3D_POINTS	This is the array of coordinates defining the 3D data points in the plot. It is an array of type Coord, where the Coord
XVW_PLOT3D_SHADETYPE	For 3D plots that are shaded, ie, when the XVW_PLOT3D_PLOTTYPE is set to one of: KPLOT3D_PHONG_SHADING KPLOT3D_GHOURAUD_SHADING or KPLOT3D_CONSTANT_SHADING, this attribute specifies what part of the data is to dictate the shading. KPLOT3D_SHADE_ELEVATION specifies that shading is to be done on elevation, ie, using the *z* value of each data point as defined by its Coord structure. KPLOT3D_SHADE_IMAGERY specifies taht shading is to be done on imagery (or color) ie, using the *d* value of each data point as defined by its Coord structure. When KPLOT3D_SHADE_NORMAL is used, the normal to each data point is computed, and shading is done using the normals.
XVW_PLOT3D_TIME_OFFSET	This attribute is used in conjunction with the XVW_PLOT3D_X_ORIENTATION and XVW_PLOT3D_Y_ORIENTATION attributes. When *neither* of these attributes is set to KTIME, it is used to specify the time offset at which the X and Y coordinates for the plot data are to be extracted.
XVW_PLOT3D_WIDTH_OFFSET	This attribute is used in conjunction with the XVW_PLOT3D_X_ORIENTATION and XVW_PLOT3D_Y_ORIENTATION attributes. When *neither* of these attributes is set to KWIDTH, it is used to specify the width offset at which the X and Y coordinates for the plot data are to be extracted.
XVW_PLOT3D_X_ORIENTATION	This attribute is only used when the plot data is provided via a kobject specified using the XVW_PLOT3D_PLOTOBJ or XVW_PLOT3D_PLOTFILE attributes; furthermore, it only applies when the plot data is stored in the value segment of the data object. It dictates how the X coordinates of the 3D plot data is to be extracted from the value segment of the data object. The 3D plot data is always extracted as a surface, but that surface may have its X coordinates oriented down width, height, depth, time, or elements.
XVW_PLOT3D_Y_ORIENTATION	This attribute is only used when the plot data is provided via a kobject specified using the XVW_PLOT3D_PLOTOBJ or XVW_PLOT3D_PLOTFILE attributes; furthermore, it only applies when the plot data is stored in the value segment of the data object. It dictates how the Y coordinates of the 3D plot data is to be extracted from the value segment of the data object. The 3D plot data is always extracted as a surface, but that surface may have its Y coordinates oriented down width, height, depth, time, or elements.

Descriptions of Plot3D Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_PLOT3D_COLOR_ORIGINATION (plot3DColorOrigination)	int	KPLOT3D_COLOR_FROM_FOREGROUND	KPLOT3D_COLOR_FROM_FOREGROUND or symbol index KPLOT3D_COLOR_FROM_DATA
XVW_PLOT3D_CONTOUR_LEVELS (plot3DContourLevels)	Real *	NULL	any array of Real values
XVW_PLOT3D_DATA_MAX_X (N/A)	double	1.0	The maximum X world coordinate value (read only)
XVW_PLOT3D_DATA_MAX_Y (N/A)	double	1.0	The maximum Y world coordinate value (read only)
XVW_PLOT3D_DATA_MAX_Z (N/A)	double	1.0	The maximum Z world coordinate value (read only)
XVW_PLOT3D_DATA_MIN_X (N/A)	double	1.0	The minimum X world coordinate value (read only)
XVW_PLOT3D_DATA_MIN_Y (N/A)	double	1.0	The minimum Y world coordinate value (read only)
XVW_PLOT3D_DATA_MIN_Z (N/A)	double	1.0	The minimum Z world coordinate value (read only)
XVW_PLOT3D_DEPTH_OFFSET (N/A)	int	0	0 < value < depth of data object value segment
XVW_PLOT3D_ELEMENTS_OFFSET (N/A)	int	0	0 < value < elements of data object value segment
XVW_PLOT3D_HEIGHT_OFFSET (N/A)	int	0	0 < value < height of data object value segment
XVW_PLOT3D_NUM_CONTOUR_LEVELS (plot3DContourLevelNum)	int	-1	value = -1, or value > 0
XVW_PLOT3D_PLOTFILE (N/A)	char *	NULL	any valid input file
XVW_PLOT3D_PLOTOBJ (N/A)	kobject	NULL	any valid data object
XVW_PLOT3D_PLOTSIZE (N/A)	int	0	values > 0
XVW_PLOT3D_PLOTTYPE (plot3DPlotType)	int	KPLOT3D_LINEPLOT	KPLOT3D_LINEPLOT KPLOT3D_MESH KPLOT3D_SCATTER KPLOT3D_CONTOUR_3D KPLOT3D_CONTOUR_2D KPLOT3D_IMPULSE KPLOT3D_PHONG_SHADING KPLOT3D_GHOURAUD_SHADING KPLOT3D_CONSTANT_SHADING KPLOT3D_HORIZON KPLOT3D_WIREFRAME

Descriptions of Plot3D Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_PLOT3D_PLOTWIDTH (N/A)	int	N/A	0 < value <= XVW_PLOT3D_PLOTSIZE
XVW_PLOT3D_POINTS (N/A)	Coord *	NULL	array of Coords defining data points for 3D plot
XVW_PLOT3D_SHADETYPE (plot3DShadeType)	int	KPLOT3D_SHADE_ELEVATION	KPLOT3D_SHADE_IMAGERY KPLOT3D_SHADE_ELEVATION KPLOT3D_SHADE_NORMAL
XVW_PLOT3D_TIME_OFFSET (N/A)	int	0	0 < value < time of data object value segment
XVW_PLOT3D_WIDTH_OFFSET (N/A)	int	0	0 < value < width of data object value segment
XVW_PLOT3D_X_ORIENTATION (N/A)	int	KWIDTH	KWIDTH KHEIGHT KDEPTH KTIME KELEMENTS
XVW_PLOT3D_Y_ORIENTATION (N/A)	int	KHEIGHT	KWIDTH KHEIGHT KDEPTH KTIME KELEMENTS

E.3. Complete Resource Set of the 3D Plot Object

The inheritance tree of the 3D plot object is as follows:

manager -> graphics -> color -> plot3D

Accordingly, the complete resource set for the 3D plot object includes:

1. The 3D plot attribute resource set, given above
2. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

E.4. Example Using the 3D Plot Visual Object

Examples of programs using the 3D Plot object can be found in `$ENVISION/examples/plot/`. One of these is as follows.

```
#include <envision.h>

/*
 * This program creates a constant shaded 3D plot from the info stored in
 * a VIFF file.
 */
void main(
   int  argc,
   char *argv[])
{
     xvobject plot3d, area;
     kobject  data_object;
     char *filename = "image:flow";

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /* open the data object defined by file */
     data_object = kpds_open_input_object(filename);

     /* create a 3D color mesh plot from the VIFF file */
     area = xvw_create_area(NULL, "area");
     xvw_set_attributes(area,
                    XVW_WIDTH,  500,
                    XVW_HEIGHT, 500,
                    NULL);

     plot3d = xvw_create_plot3d(area, "Plot3D");
     xvw_set_attributes(plot3d,
          XVW_GRAPHICS_ATTACH,  plot3d,
          XVW_PLOT3D_PLOTOBJ,   data_object,
          XVW_PLOT3D_PLOTTYPE,  KPLOT3D_MESH,
          XVW_PLOT3D_COLOR_ORIGINATION,  KPLOT3D_COLOR_FROM_DATA,
          NULL);

     /* display & run; there is no way to exit the program but ^C */
     xvf_run_form();
}
```

F. The Axis Attributes

The *xvplot* library offers a 2D axis object that can be used to put axes on 2D plots. Its three dimensional counterpart, the 3D axis object, is still under design and is not available with VisiQuest 2001. Nevertheless, 2D and 3D axis objects have a large number of attributes in common, which are referred to as general axis attributes.

These attributes are described here.

F.1. General Axis Attributes

These are general attributes for the axis.

Summary of General Axis Class Attributes	
Attribute	**Description**
XVW_AXIS_AXIS_MODE	This attribute specifies the scale with which axes will be drawn. The scale may be linear or log; natural log is planned for the future but is not implemented as of yet.
XVW_AXIS_BEGIN_AXIS	This parameter specifies where the axis begins. It is specified as a Coord structure representing a 3D point (x,y,z), where values of x, y, and z must be between 0 and 1. Default begin pt for 2D/3D axis object (X axis): (0,0,0). Default begin pt for 2D/3D axis object (Y axis): (0,0,0). Default begin pt for 3D axis object (Z axis): (0,0,0). For example, suppose we wish to prevent a 2D X axis from displaying tic marks, grid, and numerical labels above 10% of the axis range. First, the X axis is obtained from the 2D X axis object using XVW_AXIS2D_AXIS_X. Then, the XVW_AXIS_BEGIN_AXIS attribute is used to specify the begin point of the X axis as (0.1, 0.0, 0.0).
XVW_AXIS_END_AXIS	This parameter specifies where the axis ends. It is specified as a Coord structure representing a 3D point (x,y,z), where values of x, y, and z must be between 0 and 1. Default end pt for 2D/3D axis object (X axis): (1,0,0). Default end pt for 2D/3D axis object (Y axis): (0,1,0). Default end pt for 3D axis object (Z axis): (0,0,1). For example, suppose we wish to prevent a 2D X axis from displaying tic marks, grid, and numerical labels past 90% of the axis range. First, the X axis is obtained from the 2D X axis object using XVW_AXIS2D_AXIS_X. Then, the XVW_AXIS_BEGIN_AXIS attribute is used to specify the end point of the X axis as (0.9, 0.0, 0.0).
XVW_AXIS_LABEL	The text with which to label the axis.
XVW_AXIS_NUMBER_MINOR_TICS	The number of minor tics to be desired.
XVW_AXIS_TIC_JUSTIFICATION	Whether the major and minor axis tic marks appear to the outside of the axis line, to the inside of the axis line, or centered with the axis line running through the middle.

Descriptions of General Axis Class Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_AXIS_AXIS_MODE (axisAxisMode)	int	KAXIS_LINEAR	KAXIS_LINEAR KAXIS_LOG
XVW_AXIS_BEGIN_AXIS (N/A)	Coord	(0.0,0.0,0.0)	a Coord structure with values 0.0 - 1.0 specified for the (x, y, z); 2D axes will always have the z value set to 0.0.
XVW_AXIS_END_AXIS (N/A)	Coord	(1.0,1.0,1.0)	a Coord structure with values 0.0 - 1.0 specified for the (x, y, z); 2D axes will always have the z value set to 0.0.
XVW_AXIS_LABEL (N/A)	char *	"Axis"	printable text
XVW_AXIS_NUMBER_MINOR_TICS (axisNumberMinorTics)	int	0	value >= 0
XVW_AXIS_TIC_JUSTIFICATION (axisTicJustification)	int	KAXIS_INSIDE	KAXIS_INSIDE KAXIS_CENTERED KAXIS_OUTSIDE

F.2. Control of Displayed Axis Elements

An axis is made up of several components; you have control over which of these components (if any) are displayed at any given time.

Summary of Attributes For Controlling Displayed Axis Elements	
Attribute	**Description**
XVW_AXIS_SHOW_AXIS	Dictates whether or not the axis itself is displayed.
XVW_AXIS_SHOW_AXIS_LABEL	Dictates whether or not the axis label is displayed.
XVW_AXIS_SHOW_BOX	Dictates whether or not the axis box will be displayed.
XVW_AXIS_SHOW_MAJOR_GRID	Dictates whether or not a grid will be drawn using the major tics to specify location of grid lines.
XVW_AXIS_SHOW_MINOR_GRID	Dictates whether or not a grid will be drawn using the minor tics to specify location of grid lines.
XVW_AXIS_SHOW_NUMERICAL_LABELS	Dictates whether or not the numerical labels are displayed, marking the world coordinate values at the major tic marks.
XVW_AXIS_SHOW_TICS	Dictates whether or not tic marks are displayed.
XVW_AXIS_SHOW_ZERO_LINE	Dictates whether or not a line should be drawn to mark the location of zero on the axis.

Descriptions of Attributes For Controlling Displayed Axis Elements			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_AXIS_SHOW_AXIS (axisShowAxis)	int	TRUE	TRUE/FALSE
XVW_AXIS_SHOW_AXIS_LABEL (axisShowAxisLabel)	int	TRUE	TRUE/FALSE
XVW_AXIS_SHOW_BOX (axisShowBox)	int	TRUE	TRUE/FALSE
XVW_AXIS_SHOW_MAJOR_GRID (axisShowMajorGrid)	int	FALSE	TRUE/FALSE
XVW_AXIS_SHOW_MINOR_GRID (axisShowMinorGrid)	int	FALSE	TRUE/FALSE
XVW_AXIS_SHOW_NUMERICAL_LABELS (axisShowNumericalLabel)	int	TRUE	TRUE/FALSE
XVW_AXIS_SHOW_TICS (axisShowTics)	int	TRUE	TRUE/FALSE
XVW_AXIS_SHOW_ZERO_LINE (axisShowZeroLine)	int	TRUE	TRUE/FALSE

F.3. Control of Labels

A variety of attributes are available offering control over the labels of the axis.

Summary of Attributes That Control Axis Labels	
Attribute	**Description**
XVW_AXIS_LABELED_INTERVAL	The labeling interval between each major tic mark.
XVW_AXIS_LABELED_MAX	This is the largest value which is labeled on the axis; it may differ from the actual world coordinate maximum. For example, suppose that a 2D plot has X values which range from 0.0 to 0.789; it may be nicer to have the X axis labeled from 0 to 1, in which case XVW_AXIS_LABELED_MAX would be set to 1.0.
XVW_AXIS_LABELED_MIN	This is the smallest value which is labeled on the axis; it may differ from the actual world coordinate minimum. For example, suppose that a 2D plot has X values which range from 0.214 to 1.0; it may be nicer to have the X axis labeled from 0 to 1, in which case XVW_AXIS_LABELED_MIN would be set to 0.0.

Summary of Attributes That Control Axis Labels	
Attribute	**Description**
XVW_AXIS_LABELING_MODE	This attribute controls which of: (1) the label interval, (2) the number of steps, or (3) the minimum & maximum values dictate how the numbers are labeled on the axis. There are five labeling modes available: KAXIS_SET_LABELED_INTERVAL- The specified label interval is used to control labelling; the number of steps is calculated from the interval. The maximum and minimum may be changed to fit the interval. KAXIS_SET_NUMBER_STEPS- The specified number of steps is used to control labelling; the label interval is calculated from the number of steps. The maximum and minimum may be changed to fit the step number. KAXIS_SET_INTERVAL_STEPS- The specified label interval and number of steps are used. The maximum and minimum may be changed to fit the label interval and number of steps. KAXIS_SET_MAX_MIN- The specified maximum and minimum are used. The label interval will be a multiple of 1, 2, 4 and 5 and the number of steps adjusted accordingly. KAXIS_SET_DEFAULT_MODE- Similar to maximum and minimum, this is used as default mode for the initialization purposes. It differs in that the maximum and minimum will be adjusted so that they are even multiples of the label interval.

Summary of Attributes That Control Axis Labels	
Attribute	**Description**
XVW_AXIS_NICE_LABELS	When set to TRUE, labels of the axis will follow these rules: 1) Minimum will be a even multiple of the labeled interval. 2) Maximum will be a even multiple of the labeled interval and will be equal to the minimum + number of steps * label interval. 3) Each major tic label will be a even multiple of the labeled interval. When FALSE, the labels of the axis will follow these rules: 1) Minimum may or may not be a even multiple of the labeled interval. 2) Maximum may or may not be a even multiple of the labeled interval and may not even be displayed. 3) Each major tic label may or may not be a even multiple of the labeled interval. 4) If the number of steps may or may not be an even multiple of the range divided by the interval size. Note: when this attribute is set from FALSE to TRUE and back again, the labels will be re-adjusted and will not be exactly the same each time.
XVW_AXIS_NUMBER_STEPS	The number of steps to take starting at the minimum value. This parameter will control the number of major tics which is this value + 1.
XVW_AXIS_NUMERICAL_LABELS_FORMAT	This controls how the numerical labels are displayed. The format is the same as for printf(). If a numerical value is 1.34 and the format value is %g, the label will be displayed as 1.34. However, if the format is %f the label will be display as 1.340000. You can not specify string formats like %s and %c.
XVW_AXIS_RESTORE_LABELS	Setting this *action attribute* will cause the numeric labels on the axes to be restored to the actual minimum and maximum values of the world coordinates. Note that the labelled maximum and minimum could have been changed to something other than the actual minimum and maximum values with the use of the XVW_AXIS_LABELED_MIN and/or XVW_AXIS_LABELED_MAX. Thus, XVW_AXIS_RESTORE_LABELS is used to restore the original labelled values.

Descriptions of Attributes That Control Axis Labels			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_AXIS_LABELED_INTERVAL (N/A)	double	0.2	world coordinate values > 0.0
XVW_AXIS_LABELED_MAX (N/A)	double	1.0	any double world coordinate value
XVW_AXIS_LABELED_MIN (N/A)	double	0.0	any double world coordinate value

Descriptions of Attributes That Control Axis Labels			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_AXIS_LABELING_MODE (N/A)	int	KAXIS_SET_DEFAULT_MODE	KAXIS_SET_LABELED_INTERVAL KAXIS_SET_NUMBER_STEPS KAXIS_SET_INTERVAL_STEPS KAXIS_SET_MAX_MIN KAXIS_SET_DEFAULT_MODE
XVW_AXIS_NICE_LABELS (axisNiceLabels)	int	FALSE	TRUE/FALSE
XVW_AXIS_NUMBER_STEPS (N/A)	double	5.0	value > 0.0
XVW_AXIS_NUMERICAL_LABELS_FORMAT (N/A)	char *	"%g"	standard printf() formatting strings
XVW_AXIS_RESTORE_LABELS (N/A)	int	N/A	TRUE

F.4. Line Widths & Line Types

The line widths and line types are settable on the major and minor grids. Line widths range from none (invisible) to extra wide. There are seven line types available, offering a variety of possibilities for grid drawing.

Summary of Attributes That Control Grid Lines	
Attribute	**Description**
XVW_AXIS_MAJOR_GRID_LINE_TYPE	The line type with which the major grid is drawn.
XVW_AXIS_MAJOR_GRID_LINE_WIDTH	The line width with which the major grid is drawn.
XVW_AXIS_MINOR_GRID_LINE_TYPE	The line type with which the minor grid is drawn.
XVW_AXIS_MINOR_GRID_LINE_WIDTH	The line width with which the minor grid is drawn.

Descriptions of Attributes That Control Grid Lines			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_AXIS_MAJOR_GRID_LINE_TYPE (axisMajorGridLineType)	int	KLINE_GRID_DOTTED	KLINE_SOLID KLINE_DOTTED KLINE_DOT_DASH KLINE_SHORT_DASH KLINE_LONG_DASH KLINE_ODD_DASH KLINE_GRID_DOTTED

Descriptions of Attributes That Control Grid Lines			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_AXIS_MAJOR_GRID_LINE_WIDTH (axisMajorGridLineWidth)	int	KLINE_MEDIUM_FINE	KLINE_NONE KLINE_EXTRA_FINE KLINE_FINE KLINE_MEDIUM_FINE KLINE_MEDIUM KLINE_MEDIUM_WIDE KLINE_WIDE KLINE_EXTRA_WIDE
XVW_AXIS_MINOR_GRID_LINE_TYPE (axisMinorGridLineType)	int	KLINE_GRID_DOTTED	KLINE_SOLID KLINE_DOTTED KLINE_DOT_DASH KLINE_SHORT_DASH KLINE_LONG_DASH KLINE_ODD_DASH KLINE_GRID_DOTTED
XVW_AXIS_MINOR_GRID_LINE_WIDTH (axisMinorGridLineWidth)	int	KLINE_EXTRA_FINE	KLINE_NONE KLINE_EXTRA_FINE KLINE_FINE KLINE_MEDIUM_FINE KLINE_MEDIUM KLINE_MEDIUM_WIDE KLINE_WIDE KLINE_EXTRA_WIDE

F.5. Setting Colors of Axis Elements

You may set the foreground color of various parts of the axis using either the name string of a color or a pixel value. Names of valid color name strings can be found by looking at /usr/lib/X11/rgb.txt, but be aware that the presence of a color name string in this file does not guarantee that the color will be available on a particular X server. Note that all attributes involving a *color name* will be over-ridden by their counterparts that involve a *pixel value*.

Summary of Attributes For Setting Color on Axes	
Attribute	**Description**
XVW_AXIS_AXIS_COLOR	The string that specifies the name of the desired color for the axis.
XVW_AXIS_AXIS_PIXEL	The pixel value that defines the desired color for the axis.
XVW_AXIS_BOX_COLOR	The string that specifies the name of the desired color for the axis box.
XVW_AXIS_BOX_PIXEL	The pixel value that defines the desired color for the box.
XVW_AXIS_MAJOR_GRID_COLOR	The string that specifies the name of the desired color for the major grid.

Summary of Attributes For Setting Color on Axes	
Attribute	**Description**
XVW_AXIS_MAJOR_GRID_PIXEL	The pixel value that defines the desired color for the major grid.
XVW_AXIS_MINOR_GRID_COLOR	The string that specifies the name of the desired color for the minor grid.
XVW_AXIS_MINOR_GRID_PIXEL	The pixel value that defines the desired color for the minor grid.
XVW_AXIS_NUMERICAL_LABELS_COLOR	The string that specifies the name of the desired color for the numerical labels.
XVW_AXIS_NUMERICAL_LABELS_PIXEL	The pixel value that defines the desired color for the numerical labels.

Descriptions of Attributes For Setting Color on Axes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_AXIS_AXIS_COLOR (axisAxisColor)	char *	default fg color	any valid color name: see /usr/lib/X11/rgb.txt or Section 7.1.1 of *The XLib Programming Manual* by Adrian Nye
XVW_AXIS_AXIS_PIXEL (N/A)	unsigned long	default fg pixel (XtDefaultForeground)	any valid pixel value: see Section 7.3 and 7.4 of *The XLib Programming Manual* by Adrian Nye
XVW_AXIS_BOX_COLOR (axisBoxColor)	char *	default foreground color	any valid color name: see /usr/lib/X11/rgb.txt or Section 7.1.1 of *The XLib Programming Manual* by Adrian Nye
XVW_AXIS_BOX_PIXEL (N/A)	unsigned long	default fg pixel (XtDefaultForeground)	any valid pixel value: see Section 7.3 and 7.4 of *The XLib Programming Manual* by Adrian Nye
XVW_AXIS_MAJOR_GRID_COLOR (axisMajorGridColor)	char *	default foreground color	any valid color name: see /usr/lib/X11/rgb.txt or Section 7.1.1 of *The XLib Programming Manual* by Adrian Nye
XVW_AXIS_MAJOR_GRID_PIXEL (N/A)	unsigned long	default fg pixel (XtDefaultForeground)	any valid pixel value: see Section 7.3 and 7.4 of *The XLib Programming Manual* by Adrian Nye
XVW_AXIS_MINOR_GRID_COLOR (axisMinorGridColor)	char *	default foreground color	any valid color name: see /usr/lib/X11/rgb.txt or Section 7.1.1 of *The XLib Programming Manual* by Adrian Nye
XVW_AXIS_MINOR_GRID_PIXEL (N/A)	unsigned long	default fg pixel (XtDefaultForeground)	any valid pixel value: see Section 7.3 and 7.4 of *The XLib Programming Manual* by Adrian Nye

Descriptions of Attributes For Setting Color on Axes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_AXIS_NUMERICAL_LABELS_COLOR (axisNumericalLabelsColor)	char *	default foreground color	any valid color name: see /usr/lib/X11/rgb.txt or Section 7.1.1 of *The XLib Programming Manual* by Adrian Nye
XVW_AXIS_NUMERICAL_LABELS_PIXEL (N/A)	unsigned long	default fg pixel (XtDefaultFore-ground)	any valid pixel value: see Section 7.3 and 7.4 of *The XLib Programming Manual* by Adrian Nye

F.6. Setting the Scale of Axes

Axes support both linear and log scaling.

Summary of Attributes That Control Axis Mode	
Attribute	**Description**
XVW_GRAPHICS_MODE_X	Whether the X Axis is marked as linear or log 10 scale.
XVW_GRAPHICS_MODE_Y	Whether the Y Axis is marked as linear or log 10 scale.
XVW_GRAPHICS_MODE_Z	Whether the Z Axis is marked as linear or log 10 scale.

Summary of Attributes That Control Axis Mode	
Attribute	**Description**
XVW_GRAPHICS_PROPORTIONAL	When this attribute is set to KGRAPHICS_NONPROP, the scale in each axis direction is determined by the world coordinate minimums and maximums of the data. Thus, if the X values range from 0 to 1, and the Y values range from 1 to 100, and the physical length of the X and Y axes are the same, then the scale in the X direction (1) is much smaller than the scale in the Y direction (100). In contrast, when this attribute is set to KGRAPHICS_PROP_WINDOWED or KGRAPHICS_PROP_NONWINDOWED, it causes the scale in each axis direction to be the same. When a visual object is proportional, this means that the minimum and maximum values across being displayed as X and Y are found before the object is displayed. Then the minimum and maximum are set on each axis so that the scale is the same. If KGRAPHICS_PROP_WINDOWED is specified, then the center of the data is determined along each axis and the minimum and maximum values are set such that the data is in the center of the world coordinate space. If KGRAPHICS_PROP_NONWINDOWED is specified, the range of numbers covered, i.e, the minimum and maximum values are the same, by the X and Y axes are the same. The "sense of proportion" conveyed by the object is correct, but details of the object may be lost if the range spanned by X varies greatly from that spanned by Y. In some cases, proportional display not be a good method of display. For example, suppose a 2D plot object has data displayed as X ranges from 0 to 255, but the data displayed as Y ranges from 0 to 1. Obviously, the plot will not be very informative, as all the points will be bunched against the X axis. In cases like this, non-proportional display may be more useful.

Descriptions of Attributes That Control Axis Mode			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_GRAPHICS_MODE_X (graphicsModeX)	int	KGRAPHICS_LINEAR	KGRAPHICS_LINEAR KGRAPHICS_LOG10
XVW_GRAPHICS_MODE_Y (graphicsModeY)	int	KGRAPHICS_LINEAR	KGRAPHICS_LINEAR KGRAPHICS_LOG10
XVW_GRAPHICS_MODE_Z (graphicsModeZ)	int	KGRAPHICS_LINEAR	KGRAPHICS_LINEAR KGRAPHICS_LOG10
XVW_GRAPHICS_PROPORTIONAL (graphicsProportional)	int	KGRAPHICS_NONPROP	KGRAPHICS_NONPROP KGRAPHICS_PROP_WINDOWED KGRAPHICS_PROP_NONWINDOWED

G. The 2D Axis Object

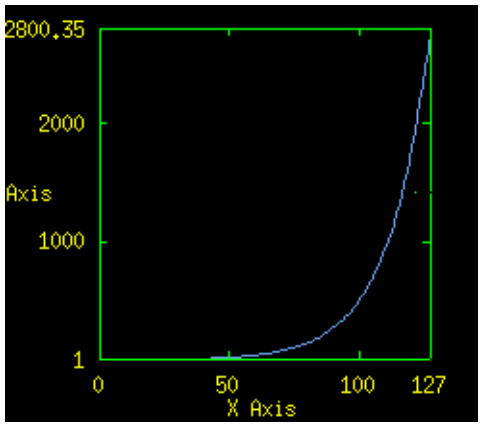


Figure 4: The 2D Axis object provides an axis system for one or more 2D plots.

G.1. xvw_create_axis2d() — *create a 2D axis object*

Synopsis

```
xvobject xvw_create_axis2d(
    xvobject parent,
    char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the 2D axis object (for use in app-defaults files, etc)

Returns

The 2D axis object on success, NULL on failure

Description

The 2D axis object provides a set of X, Y axes for use with one or more 2D plot objects. In general, the 2D axis object should be set as the visual object which controls the world view of its siblings; specify the 2D axis object as the controlling visual object when setting the XVW_GRAPHICS_ATTACH attributes on its siblings.

G.2. Attributes of the Axis2D Visual Object

Summary of Axis2D Attributes	
Attribute	**Description**
XVW_AXIS2D_AXIS_X	This *read-only* attribute allows you to obtain the axis object that serves as the X Axis.
XVW_AXIS2D_AXIS_Y	This *read-only* attribute allows you to obtain the axis object that serves as the Y Axis
XVW_AXIS2D_SHOW_AXIS_X	The X axis is displayed when this attribute is set to TRUE; it is not displayed when this attribute is set to FALSE.
XVW_AXIS2D_SHOW_AXIS_Y	The Y axis is displayed when this attribute is set to TRUE; it is not displayed when this attribute is set to FALSE.

Descriptions of Axis2D Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_AXIS2D_AXIS_X (N/A)	xvobject	NULL	The axis object that is used as the X axis (read only).
XVW_AXIS2D_AXIS_Y (N/A)	xvobject	NULL	The axis object that is used as the Y axis (read only).
XVW_AXIS2D_SHOW_AXIS_X (axis2dShowAxisX)	int	TRUE	TRUE/FALSE
XVW_AXIS2D_SHOW_AXIS_Y (axis2dShowAxisY)	int	TRUE	TRUE/FALSE

G.3. Complete Resource Set of the Axis2D Visual Object

The inheritance tree of the animation object is as follows:

manager -> graphics -> axis2D

Accordingly, the complete resource set for the 2D axis object includes:

1. The 2D axis attribute resource set, given above
2. The general axis attribute resource set, given in Section F, "The Axis Attributes"
3. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
4. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

G.4. Example Using the Axis2D Visual Object

Examples of programs using the 2D Axis object can be found in `$ENVISION/examples/plot/`. One of these is as follows.

```
#include <envision.h>

Coord  data[4000];

void  recompute_points PROTO((xvobject, kaddr, XEvent *, int *));

/*
 * This example demonstrates how an area object can be used to provide
 * a coordinated world coordinate system for a plot and a set of axes.
 * It creates an area object, with a 2D plot and a set of 2D axes within
 * it;  it attaches the plot and the axes to the area object, so they
 * have a coordinated world coordinate view, and keep that coordinated
 * view even when the range of the plot changes.
 *
 * Click on the image to invoke the event handler which re-computes the
 * sin curve plotted to a new range;  the plot and the axis system will
 * update together.
 */
#define PTNUM 50
void main(
   int  argc,
   char *argv[])
{
     xvobject parent, area, axis, plot;
     int    i, degrees;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

     /* initialize xvwidgets lib */
        if (!xvw_initialize(XVW_MENUS_XVFORMS))
        {
        kerror(NULL, "main", "Cannot open display");
        kexit(KEXIT_FAILURE);
        }

     /*
      *  Generate initial sin curve to be plotted
      */
     for (i = 0, degrees = 9; i < PTNUM; i++, degrees+=20)
     {
        data[i].x = i;
        data[i].y = sin((double) kdegrees_radians(degrees));
        data[i].d = KGRAPHICS_UNINITIALIZED;
     }

     /*
      * create a parent for the area
      */
     parent = xvw_create_manager(NULL, "parent");
     xvw_set_attributes(parent,
                  XVW_WIDTH,  400,
                  XVW_HEIGHT, 400,
                  NULL);
```

```
        /*
         * create a coordinated axis and plot object in an area object
         */
        area = xvw_create_area(parent, "area");
        xvw_set_attributes(area,
                              XVW_AREA_DISPLAY_DATE,  FALSE,
                              XVW_AREA_TITLE_STRING, "2D Plot",
                        XVW_TACK_EDGE,          KMANAGER_TACK_ALL,
                              XVW_BACKGROUND_COLOR,   "black",
                              NULL );

        /*
         *  create the axis system
         */
        axis = xvw_create_axis2d(area, "axis");
        xvw_set_attributes(axis,
                     XVW_GRAPHICS_ATTACH, axis,
                        XVW_GRAPHICS_VIEWPORT_MIN_X, 0.2,
                              XVW_GRAPHICS_VIEWPORT_MIN_Y, 0.2,
                              XVW_GRAPHICS_VIEWPORT_MAX_X, 0.9,
                              XVW_GRAPHICS_VIEWPORT_MAX_Y, 0.9,
                     NULL);


        /*
         * The viewport max and mins do not have to be set for the plot
         * object, since it's attached to the axis object. They will have the
         * same values as the axis object, since the parent of the axis
            * object is an area object.  This is also true for the world
            * coordinates; however it is the plot that is dictating these values
            * (i.e. the maximum and minimum of the data being plotted) and the
            * axis that is inheriting them.
         */
        plot = xvw_create_plot2d(area, "plot coordinated");
        xvw_set_attributes(plot,
                     XVW_GRAPHICS_ATTACH,   axis,
                     XVW_PLOT2D_POINTS,       data,
                        XVW_PLOT2D_PLOTSIZE,       PTNUM,
                              XVW_FOREGROUND_COLOR,  "Magenta",
                        NULL);

           xvw_add_event(area, ButtonPressMask, recompute_points, plot);

        xvf_run_form();
    }

    void  recompute_points(
        xvobject object,
        kaddr    client_data,
        XEvent   *event,
        int      *dispatch)
    {
           int        i, degrees;
           static int factor = 2;
        xvobject   plot_object = (xvobject) client_data;

           /*
            *  Generate a new sin curve to be plotted
            */
```

```
        for (i = 0, degrees = 9; i < (PTNUM * factor); i++, degrees+=20)
        {
            data[i].x = i;
            data[i].y = sin((double) kdegrees_radians(degrees));
         data[i].d = KGRAPHICS_UNINITIALIZED;
        }
        xvw_set_attributes(plot_object,
                    XVW_PLOT2D_POINTS,          data,
                    XVW_PLOT2D_PLOTSIZE,        (PTNUM * factor),
                    NULL);
        factor++;
        if ( factor > 5 ) factor = 1;
}
```

H. The Indicator Object

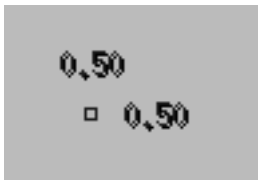


Figure 5: The indicator object simply provides a marker for a world coordinate position; the X and Y values of that position are displayed.

H.1. xvw_create_indicator() — *creates an indicator object*

Synopsis

```
xvobject xvw_create_indicator(
    xvobject parent,
    char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the indicator object (for use in app-defaults files, etc)

Returns

The indicator object on success, NULL on failure

Description

An indicator is a marker with the additional capability of displaying the X and/or Y world coordinates of the location at which the marker is placed. If desired, the indicator may also include a vertical line (most often used with displaying X values) and/or a horizontal line (most often used with displaying Y values). An indicator object is often used in conjunction with a 2D axis object to highlight a specific data point.

H.2. Attributes of the Indicator Visual Object

Summary of Indicator Attributes

Attribute	Description
XVW_INDICATOR_CONSTRAINT	Indicates whether x and/or y values are to be displayed along with the indicator. It is set specifying one of the allowed values or by OR'ing the x and y values to do both. The KINDICATOR_CONSTRAINT_X value is used to specify that the indicator should be constrained in the horizontal direction; which means that no X position indication will be given. The KINDICATOR_CONSTRAINT_Y value is used to specify that the indicator should be constrained in the vertical direction; which means that no Y position indication will be given. KINDICATOR_CONSTRAINT_NONE, the default value, specifies that both vertical and horizontal indication will be given.
XVW_INDICATOR_LINE	This attribute indicates whether a x and/or y line should be displayed along with the indicator. It is set specifying one of the allowed values or by OR'ing the x and y values to do both. The KINDICATOR_HORIZONTAL value specifies that a horizontal line should be displayed, spanning the width of the parent object. The KINDICATOR_VERTICAL value specifies that a vertical line should be displayed, spanning the height of the parent object.
XVW_INDICATOR_SHOW_XPOS	Indicates whether the indicator should display its X position.
XVW_INDICATOR_SHOW_YPOS	Indicates whether the indicator should display its Y position.
XVW_INDICATOR_XPOS	This read-only attribute returns the stringvalue visual object that displays the X position of the indicator.If the XVW_INDICATOR_SHOW_YPOS attribute is set to FALSE, the returned xvobject will be NULL.
XVW_INDICATOR_XPOS_VALUE	This attribute is the x position of the indicator in world coordinates.
XVW_INDICATOR_YPOS	This read-only attribute returns the stringvalue visual object that displays the Y position of the indicator. If the XVW_INDICATOR_SHOW_XPOS attribute is set to FALSE, the returned xvobject will be NULL.
XVW_INDICATOR_YPOS_VALUE	This attribute is the y position of the indicator in world coordinates.

Descriptions of Indicator Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_INDICATOR_CONSTRAINT (indicatorConstraint)	int	KINDICATOR_CONSTRAINT_NONE	KINDICATOR_CONSTRAINT_NONE KINDICATOR_CONSTRAINT_X KINDICATOR_CONSTRAINT_Y
XVW_INDICATOR_LINE (indicatorLine)	int	KINDICATOR_LINE_NONE	KINDICATOR_LINE_NONE KINDICATOR_LINE_VERTICAL KINDICATOR_LINE_HORIZONTAL KINDICATOR_LINE_BOTH
XVW_INDICATOR_SHOW_XPOS (indicatorShowXpos)	int	TRUE	TRUE/FALSE
XVW_INDICATOR_SHOW_YPOS (indicatorShowYpos)	int	TRUE	TRUE/FALSE
XVW_INDICATOR_XPOS (N/A)	xvobject	N/A (read only)	string xvobject
XVW_INDICATOR_XPOS_VALUE (N/A)	double	KMAXFLOAT	any double world coordinate value
XVW_INDICATOR_YPOS (N/A)	xvobject	N/A (read only)	string xvobject
XVW_INDICATOR_YPOS_VALUE (N/A)	double	KMAXFLOAT	any double world coordinate value

H.3. Complete Resource Set of the Indicator Visual Object

The inheritance tree of the indicator object is as follows:

manager -> graphics -> marker -> indicator

Accordingly, the complete resource set for the indicator object includes:

1. The indicator attribute resource set, given above
2. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

H.4. Example using the Indicator Visual Object

An example program using the indicator visual object can be found in `$ENVISION/examples/annotate/03.indicator`. This program is as follows.

```
#include <envision.h>
```

```
/*
 * This program creates a window with an area object, containing a
 * 2D axis object and an indicator object.  The indicator object displays
 * its world coordinate position within the area object. The internal menuform
 * for the indicator is displayed automatically, and may be used to set
 * attributes of the indicator.
 *
 * The indicator allows you to interactively it by "grabbing" it with the
 * left mouse button and moving it around inside the area object; it will
 * update the world coordinate value that it displays as it is moved.
 * The 2D axis object serves to provide a reference as to the world
 * coordinate system.
 */

void main(
   int  argc,
   char *argv[])
{
        xvobject parent, area, axis, indicator;

     /* initialize khoros program */
     khoros_initialize(argc, argv, "ENVISION");

     /* initialize the xvwidgets lib */
        if (!xvw_initialize(XVW_MENUS_XVFORMS))
        {
           kerror(NULL, "main", "unable to open display");
           kexit(KEXIT_FAILURE);
        }

     /*
      *  create a manager backplane
      */
     parent = xvw_create_manager(NULL, "parent");

        /*
         * create an area object to provide a coordinated WC view
      * for the 2D axis object and the indicator object
         */
        area = xvw_create_area(parent, "area");
        xvw_set_attributes(area,
                     XVW_AREA_DISPLAY_DATE,     FALSE,
                     XVW_AREA_DISPLAY_TITLE,    TRUE,
               XVW_MAXIMUM_WIDTH,                 300,
               XVW_MAXIMUM_HEIGHT,                300,
                     NULL );

        /*
         *  the 2D axis system will display the world coordinates
         */
        axis = xvw_create_axis2d(area, "axis coordinated");
        xvw_set_attributes(axis,
                     XVW_GRAPHICS_ATTACH,         axis,
                     XVW_GRAPHICS_VIEWPORT_MIN_X, 0.1,
                     XVW_GRAPHICS_VIEWPORT_MIN_Y, 0.1,
                     XVW_GRAPHICS_VIEWPORT_MAX_X, 0.9,
                     XVW_GRAPHICS_VIEWPORT_MAX_Y, 0.9,
                     NULL);
```

```
        /*
         *  create the indicator object;  attach it to the axis so
         *  that the axis will control the world view.  set the X & Y
         *  position values, and set the foreground color.
         */
           indicator = xvw_create_indicator(area, "indicator");
           xvw_set_attributes(indicator,
                   XVW_GRAPHICS_ATTACH,      axis,
             XVW_INDICATOR_XPOS_VALUE, 0.5,
             XVW_INDICATOR_YPOS_VALUE, 0.5,
                XVW_FOREGROUND_COLOR,     "pink",
             NULL);

        /* display internal menuform for indicator */
        xvw_activate_menu(indicator);

        /* display and run the program */
           xvf_run_form();
}
```

Table of Contents

This page left intentionally blank

Chapter 7

Xvannotate

Chapter 7 - Xvannotate

A. Overview of Visual Objects Related To Annotation

A variety of annotations are supported by the *xvannotate* library. Text, lines, circles, rectangles, polylines, and markers all have corresponding visual objects which support the annotation drawing needs of most applications. With some limitations, annotations may be interactively created, moved, and resized. The visual objects serving as annotations are:

- the circle object
- the date object
- the ellipse object
- the indicator object
- the line object
- the marker object
- the polyline object
- the rectangle object
- the string object
- the stringvalue object
- the timer object

B. Issues Related to Annotations

The issues related to support of annotations by GUI & Visualization services are extensive. First, each annotation must be complete and support full functionality "on its own"; by this, we mean that each class-specific resource of each annotation is supported and debugged. Secondly, annotations should support placement and size specification in both device coordinates and world coordinates (although not necessarily both at the same time). The option of specifying size and location of annotations in either device coordinates or world coordinates provides the flexibility that allows annotations to be used easily by a wide variety of applications.

Another important requirement is that annotations in general must must work correctly in conjunction with the other visual objects on which they are created; in other words, they must support an *integrated world coordinate view*. For example, suppose that a marker is created as a child of an image object which uses a pan icon, and that the marker is placed using world coordinates. If the image object is displaying a large image, and the pan icon is used to move the portion of the image that appears in the image window, the marker should move with the image, thus maintaining its integrity with respect to the world coordinate view imposed by its image parent.

Furthermore, annotations should smoothly support interactive resizing and movement. In keeping with VisiQuest 2001 GUI & Visualization services philosophy, if the parent of the annotation is put into "edit mode", the user should be able to easily select, move, and resize the annotation in a way which is intuitive and easy to use.

Another issue involves "sharing" of annotations between visual objects such that more than one visual object can have knowledge of an annotation. For example, suppose that a circle annotation is created as the child of an image; suppose further that a zoom object is used in conjunction with the image object. In this case, it would be desirable for the zoom object to display the circle annotation when its focus was placed over the area of the image where the circle object was located; in order to do this, it is necessary for the zoom and image objects to "share" knowledge of the existance of the circle annotation.

The saving and restoring of annotations is also an issue; once created, it would be nice to be able to save annotations along with the data being displayed by the parent on which they were created. Along similar lines, it has also been requested that GUI & Visualization services support quality PostScript output of annotations (along with requests of similar support for most or all other visual objects). Reflecting these issues, the VisiQuest 2 plan for annotations can be summarized briefly in the following list:

1. General infrastructure support needed for implementation of annotations (done)

2. Support for size and location specification in both device and world coordinates (done)

3. Correct operation in conjunction with other visual objects (done) (integrated world coordinate view)

4. Support for smooth interactive resize and movement by the user (work begun)

5. "Sharing" of annotations between image-subclassed objects

6. Saving and restoring of annotations

7. Quality PostScript output of annotations

Not surprisingly, the plans for VisiQuest 2001 annotations have been more extensive than the time allowed for implementation. As of this release, the three items in the plan for annotations have been completed. The infrastructure to support annotations is firmly in place, annotations can be sized and placed in either device or world coordinates (with a couple exceptions), and annotations maintain an integrated world coordinate view with other visual objects. In addition, some work has been done to support smooth interactive resize and movement by the user, although there is still a great deal of work left in this area.

As of this release, annotations cannot be "shared" between image-subclassed objects as mentioned above. There is no provision for saving or restoring of annotations, and, unfortunately, no support for quality PostScript output. Hopefully these additional items will be addressed in upcoming VisiQuest 2001 releases. The table below gives a summary of the current status of the different annotations provided by the *xvannotate* library:

Annotions				
Annotation	**Device Coordinate Size/Position**	**World Coordinate Size/Position**	**Interactive Movement**	**Interactive Resize**

Annotions				
Annotation	**Device Coordinate Size/Position**	**World Coordinate Size/Position**	**Interactive Movement**	**Interactive Resize**
Circle	Yes	Yes	Easy	Sometimes Difficult
Date	Yes	Yes	Easy	N/A
Ellipse	Yes	Yes	Easy	Sometimes Difficult
Indicator	Yes	Yes	Very Difficult	N/A
Line	Awkward	Yes	Difficult to select; motion limited to 90 degrees; sometimes unpredictable behavior may leave mouse droppings	Difficult to select; Sometimes unpredictable behavior may leave mouse droppings
Marker	Yes	Yes	Easy	N/A
Polyline	No	Yes	Not supported yet	Not supported yet
Rectangle	Yes	Yes	Easy, but leaves mouse droppings	Easy, but leaves mouse droppings
String	Yes	Yes	Easy	N/A
StringValue	Yes	Yes	Easy	N/A
Timer	Yes	Yes	Easy	N/A

C. The Circle Object

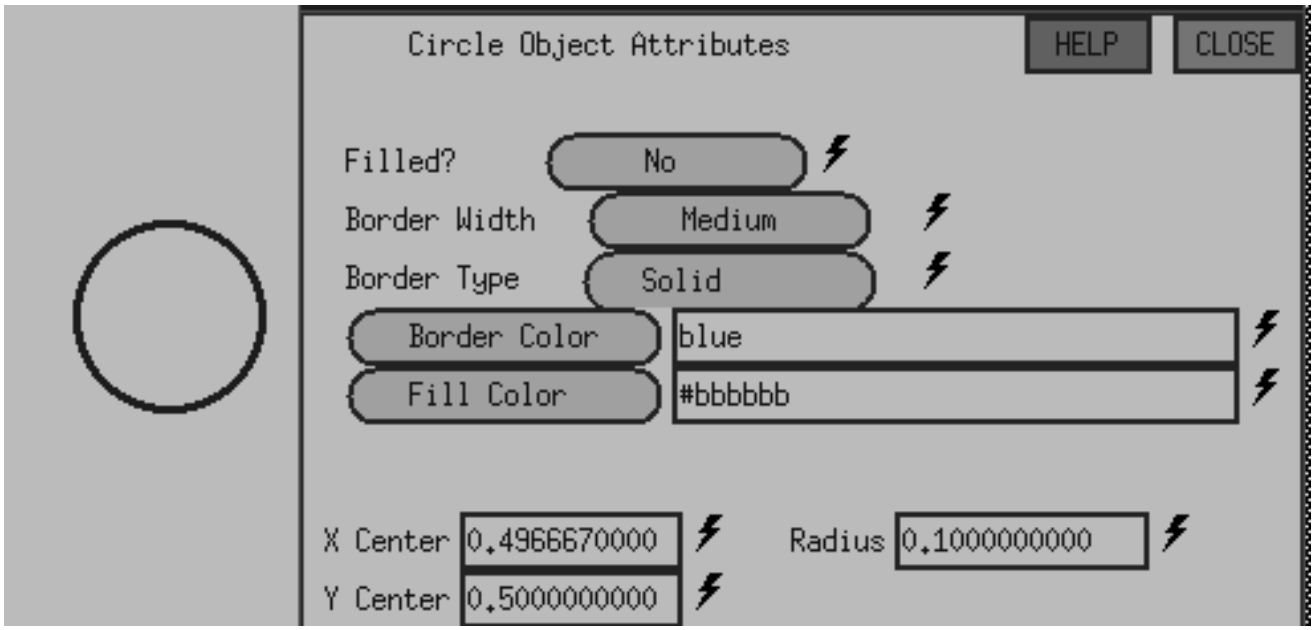


Figure 1: The circle object with its internal menuform displayed.

C.1. xvw_create_circle() — *create a circle object.*

Synopsis

```
xvobject xvw_create_circle(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically.

`name`
: a name for this particular instance of the circle object (for use in app-defaults files, etc)

Returns

The circle xvobject on success, NULL on failure

Description

A circle visual object supports the display of a circle.

The (x, y) location of the circle center and the radius of the circle may be specified in world coordinates, where the world coordinate range is dictated by the "controlling" visual object. By default, the parent of the circle object is its controlling visual object; this may be changed using the

XVW_GRAPHICS_ATTACH attribute.

Alternatively, the size and location of the circle may be specified using device coordinates. When device coordinates are used, they are with respect to the bounding box surrounding the circle; specify the (x,y) location of the upper left hand corner of the bounding box, as well as the width and height of the bounding box.

C.2. Attributes of the Circle Visual Object

Summary of Circle Attributes	
Attribute	**Description**
XVW_CIRCLE_RADIUS	This double value specifies the radius of the circle in world coordinates.
XVW_CIRCLE_XCENTER	This double value specifies the x location of the center of the circle in world coordinates.
XVW_CIRCLE_YCENTER	This double value specifies the y location of the center of the circle in world coordinates.

Descriptions of Circle Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_CIRCLE_RADIUS (N/A)	double	1.0	+/- double
XVW_CIRCLE_XCENTER (N/A)	double	0.0	+/- double
XVW_CIRCLE_YCENTER (N/A)	double	0.0	+/- double

C.3. Complete Resource Set of the Circle Visual Object

The inheritance tree of the circle object is as follows:

manager -> graphics -> circle

Accordingly, the complete resource set for the circle object includes:

1. The circle attribute resource set, given above
2. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

C.4. Example using the Circle Visual Object

An example program using the circle visual object can be found in `$ENVISION/examples/annotate/06.circle`. This program is as follows.

```
#include <envision.h>

/*
 * This example creates a window with a single circle annotation in it.
 *
 * After the manager object parent is put into edit mode (hold down
 * shift key and click the left mouse button), the circle can be
 * interactively moved by "grabbing" it in the middle with the left mouse
 * button; grabbing it near an end will cause the circle to resize.
 *
 * In this example, the internal menuform for the circle is displayed
 * automatically.
 */

void main(
   int  argc,
   char *argv[])
{
        xvobject parent;
        xvobject circle;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

        /* initialize the xvwidgets lib */
        if (!xvw_initialize(XVW_MENUS_XVFORMS))
        {
           kerror(NULL, "main", "unable to open display");
           kexit(KEXIT_FAILURE);
        }

        /*
         *  create black manager backplane, width & height of 300.
         */
        parent = xvw_create_manager(NULL, "parent");
        xvw_set_attributes(parent,
                           XVW_WIDTH,            300,
                           XVW_HEIGHT,           300,
                           XVW_BACKGROUND_COLOR, "black",
                           NULL);

     /*
      *  create the circle object.  specify dimensions in world coordinates,
      */
        circle = xvw_create_circle(parent, "circle");
        xvw_set_attributes(circle,
                           XVW_CIRCLE_XCENTER, 0.5,
                           XVW_CIRCLE_YCENTER, 0.5,
```

```
                            XVW_CIRCLE_RADIUS,  0.1,
                            XVW_FOREGROUND_COLOR, "orange",
                            NULL);
        /*
         *  activate menuform so user doesn't have to bring it up.
         */
        xvw_activate_menu(circle);

        /* display & run the program */
           xvf_run_form();
}
```

D. The Date Object

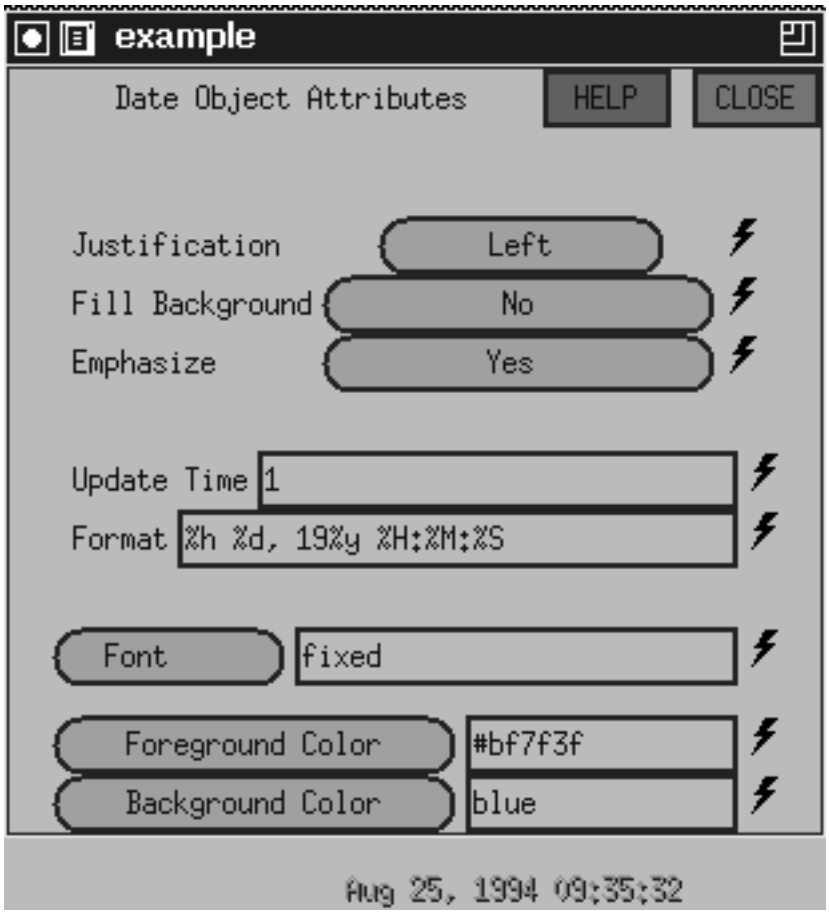


Figure 2: The date object with its internal menuform displayed. The date object is used by xprism to display the date when the plot is created.

D.1. xvw_create_date() — *create a date object.*

Synopsis

```
xvobject xvw_create_date(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the date object (for use in app-defaults files, etc)

Returns

The date object on success, NULL on failure

Description

A date visual object displays the current date according to the internal clock of your computer.

The location of the upper left corner of the the text displaying the date may be specified in world coordinates, where the world coordinate range is dictated by the "controlling" visual object. By default, the parent of the date object is its controlling visual object; this may be changed using the XVW_GRAPHICS_ATTACH attribute.

Alternatively, the location of the date may be specified using device coordinates. As when world coordinates are used, specify the (x,y) location of the upper left hand corner of the text that will display the date.

The amount of space used by the text displaying the date cannot be explicitly set by the application; this is automatically calculated according to the font that is used. Specification of a width and height simply provides a buffer of space around the text displaying the date; note that justification will have no effect unless the width and height are specified to be larger than what is actually needed by the text displaying the date. Width and height may be specified in characters or in pixels.

In addition to font specification, the date object also supports multiple styles in which the string displaying the date can appear: available styles include plain, emphasized, embossed in, and embossed out.

D.2. Attributes of the Date Visual Object

Summary of Date Attributes	
Attribute	**Description**
XVW_DATE_FORMAT	The format in which the date is displayed. Formatting strings may use any of the following field descriptors, which are taken directly from the strftime man page (`% man strftime` for more information): %% same as % %a day of week, using locale's abbreviated weekday names %A day of week, using locale's full weekday names %b month, using locale's abbreviated month names %B month, using locale's full month names %c date and time as %x %X %C date and time, in locale's long-format representation %d day of month (01-31) %D date as %m/%d/%y %e day of month (1-31; single digits are preceded by a blank) %H hour (00-23) %I hour (00-12) %j day number of year (001-366) %k hour (0-23; single digits are preceded by a blank) %l hour (1-12; single digits are preceded by a blank) %m month number (01-12) %M minutes (0-59) %n carriage return %p locale's equivalent of AM or PM %r time as %I:%M:%S %p %R time as %H:%M %S seconds (00-59) %t tab %T time as %H:%M:%S %U week number of year (01-52), Sunday is the first day of the week %w day of week; Sunday is day 0 %W week number of year (01-52), Monday is the first day of the week %x date, using locale's date format %X time, using locale's time format %y year within century (00-99) %Y year, including century (for example, 1988) %Z time zone abbreviation The difference between %U and %W lies in which day is counted as the first day of the week. Week number 01 is the first week with four or more January days in it.
XVW_DATE_UPDATETIME	This integer value specifies how often, in seconds, that the date is updated. The default is to update the date every second.

Descriptions of Date Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_DATE_FORMAT (dateFormat)	char *	%h %d, 19%y %H:%M	see description
XVW_DATE_UPDATETIME (dateUpdatetime)	int	1	values > 0

D.3. Complete Resource Set of the Date Visual Object

The inheritance tree of the date object is as follows:

manager -> graphics -> string -> date

Accordingly, the complete resource set for the date object includes:

1. The date attribute resource set, given above
2. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

D.4. Example using the Date Visual Object

An example program using the date visual object can be found in `$ENVISION/examples/annotate/11.date`. This program is as follows.

```
#include <envision.h>

/*
 * This program creates a window with a single date annotation in it.
 * The date annotation displays the current time and date.
 *
 * The date object may be selected and moved by holding down the shift key and
 * using the left mouse button; it may be resized by holding down the shift key
 * and holding the left mouse button while the mouse is on one of the boxes
 * around the bounds of the text.
 *
 * The internal menuform for the date annotation is popped up explicitly in this
 * example, but it may also be brought up by holding down the meta key and using
 * the middle mouse button.
 */

void main(int  argc,
   char *argv[])
{
        xvobject date;
```

```
    xvobject manager;

       /* initialize VisiQuest program */
       khoros_initialize(argc, argv, "ENVISION");

    /* initialize the xvwidgets lib */
       if (!xvw_initialize(XVW_MENUS_XVFORMS))
       {
          kerror(NULL, "main", "unable to open display");
          kexit(KEXIT_FAILURE);
       }

    /*
     *  create a manager backplane, specify character width and height
     */
    manager = xvw_create_manager(NULL, "backplane");
    xvw_set_attributes(manager,
                 XVW_CHAR_WIDTH,  50.0,
                 XVW_CHAR_HEIGHT,  2.0,
                 NULL);

    /*
     *  create date object specifying placement in world coordinates,
        *  the format for printing the date, and the update time.
     */
       date = xvw_create_date(manager, "date");
       xvw_set_attributes(date,
                 XVW_STRING_XPLACEMENT, 0.5,
                 XVW_STRING_XPLACEMENT, 0.5,
                 XVW_DATE_FORMAT,   "%h %d, 19%y %H:%M:%S",
                 XVW_DATE_UPDATETIME,    1,
                 NULL);

    /*
     *  activate menuform so user doesn't have to meta-click to bring it up.
     */
    xvw_activate_menu(date);

    /* display & run the program */
       xvf_run_form();
}
```

E. The Ellipse Object

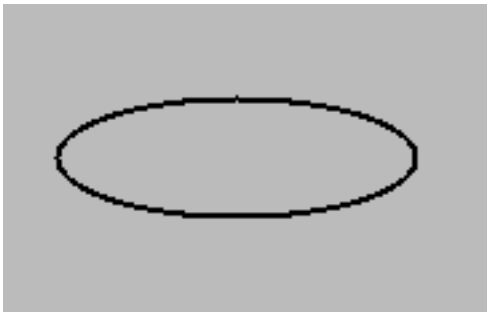


Figure 3: The ellipse annotation.

E.1. xvw_create_ellipse() — *create a ellipse object.*

Synopsis

```
xvobject xvw_create_ellipse(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
the parent object; NULL will cause a default toplevel to be created automatically.

`name`
a name for this particular instance of the ellipse object (for use in app-defaults files, etc)

Returns

The ellipse xvobject on success, NULL on failure

Description

An ellipse visual object supports the display of an ellipse.

The (x, y) location of the ellipse center, its width and height may be specified in world coordinates, where the world coordinate range is dictated by the "controlling" visual object. By default, the parent of the ellipse object is its controlling visual object; this may be changed using the XVW_GRAPHICS_ATTACH attribute.

Alternatively, the size and location of the ellipse may be specified using device coordinates. When device coordinates are used, they are with respect to the bounding box surrounding the ellipse; specify the (x,y) location of the upper left hand corner of the bounding box, as well as the width and height of the bounding box.

E.2. Attributes of the Ellipse Visual Object

Summary of Ellipse Attributes	
Attribute	**Description**
`XVW_ELLIPSE_HEIGHT`	This double value specifies the height of the ellipse in world coordinates.

Summary of Ellipse Attributes	
Attribute	**Description**
XVW_ELLIPSE_WIDTH	This double value specifies the width of the ellipse in world coordinates.
XVW_ELLIPSE_XCENTER	This double value specifies the x location of the center of the ellipse in world coordinates.
XVW_ELLIPSE_YCENTER	This double value specifies the y location of the center of the ellipse in world coordinates.

Descriptions of Ellipse Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_ELLIPSE_HEIGHT (N/A)	double	0.75	+/- double
XVW_ELLIPSE_WIDTH (N/A)	double	1.0	+/- double
XVW_ELLIPSE_XCENTER (N/A)	double	0.0	+/- double
XVW_ELLIPSE_YCENTER (N/A)	double	0.0	+/- double

E.3. Complete Resource Set of the Ellipse Visual Object

The inheritance tree of the ellipse object is as follows:

manager -> graphics -> ellipse

Accordingly, the complete resource set for the ellipse object includes:

1. The ellipse attribute resource set, given above
2. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

E.4. Example using the Ellipse Visual Object

An example program using the ellipse visual object can be found in `$ENVISION/examples/anno-`

`tate/06.ellipse`. This program is as follows.

F. The Line Object

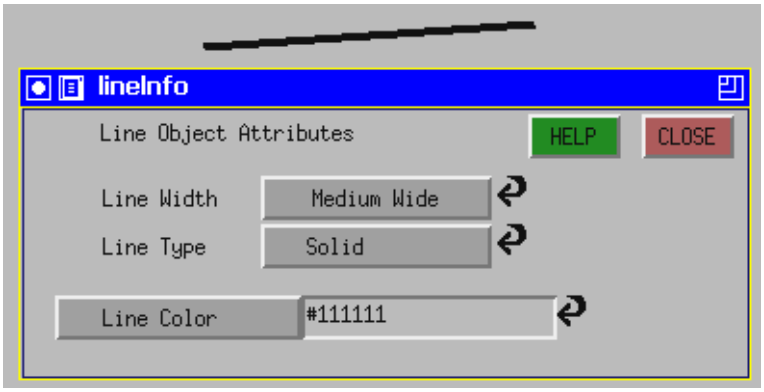


Figure 4: The line object with its internal menuform displayed. The line object is used as an annotation in a variety of applications.

F.1. xvw_create_line() — *create a line object.*

Synopsis

```
xvobject xvw_create_line(
   xvobject parent,
   char   *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the line object (for use in app-defaults files, etc)

Returns

The line object on success, NULL on failure

Description

A line visual object supports the display of a line.

The (x, y) location of the two end points of the line may be specified in world coordinates, where the world coordinate range is dictated by the "controlling" visual object. By default, the parent of the line object is its controlling visual object; this may be changed using the XVW_GRAPHICS_ATTACH

attribute.

While not recommended, the size and location of the line may be specified using device coordinates. When device coordinates are used, they are with respect to the bounding box surrounding the line; specify the (x,y) location of the upper left hand corner of the bounding box, as well as the width and height of the bounding box. Due to lack of directional information, the line in this case will be drawn from the upper left hand corner of the bounding box to the lower right of the bounding box.

F.2. Attributes of the Line Visual Object

Summary of Line Attributes	
Attribute	**Description**
XVW_LINE_XBEGIN	This double value specifies the x location of the beginning of the line segment in world coordinates.
XVW_LINE_XEND	This double value specifies the x location of the end of the line segment in world coordinates.
XVW_LINE_YBEGIN	This double value specifies the y location of the beginning of the line segment in world coordinates.
XVW_LINE_YEND	This double value specifies the y location of the end of the line segment in world coordinates.

Descriptions of Line Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_LINE_XBEGIN (N/A)	double	0.0	any double world coordinate value
XVW_LINE_XEND (N/A)	double	1.0	any double world coordinate value
XVW_LINE_YBEGIN (N/A)	double	0.0	any double world coordinate value
XVW_LINE_YEND (N/A)	double	1.0	any double world coordinate value

F.3. Complete Resource Set of the Line Visual Object

The inheritance tree of the line object is as follows:

manager -> graphics -> line

Accordingly, the complete resource set for the line object includes:

1. The line attribute resource set, given above

2. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"

3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

F.4. Example using the Line Visual Object

An example program using the line visual object can be found in `$ENVISION/examples/annotate/05.line/`. This program is as follows.

```
#include <envision.h>

/*
 * This example creates a window with a single line annotation in it.
 *
 * After the manager object parent is put into edit mode (hold down
 * shift key and click the left mouse button), the line can be
 * interactively moved by "grabbing" it in the middle with the left mouse
 * button; grabbing it near an end will cause the line to resize.
 *
 * The internal menuform for the line may be displayed by clicking
 * the middle mouse button on the line; the menuform may then be
 * used to set attributes of the line.
 */

void main(
   int  argc,
   char *argv[])
{
        xvobject line;
        xvobject image;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

     /* initialize the xvwidgets lib */
        if (!xvw_initialize(XVW_MENUS_XVFORMS))
        {
           kerror(NULL, "main", "unable to open display");
           kexit(KEXIT_FAILURE);
        }

     /*
      * create image object to serve as backplane for line annotation
      */
     image = xvw_create_image(NULL, "image");
     xvw_set_attribute(image, XVW_IMAGE_IMAGEFILE, "image:kitten");

     /*
      *  create red line object; specify endpoints in world coordinates.
```

```
        */
          line = xvw_create_line(image, "line");
          xvw_set_attributes(line,
                             XVW_LINE_XBEGIN, 45.0,
                             XVW_LINE_YBEGIN, 145.0,
                             XVW_LINE_XEND,   100.0,
                             XVW_LINE_YEND,   155.0,
                       XVW_FOREGROUND_COLOR, "red",
                             NULL);
      /* display & run the program */
          xvf_run_form();
}
```

G. The Marker Object

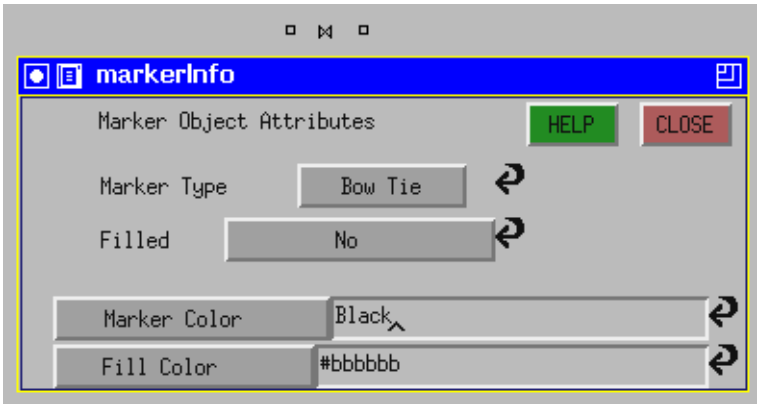


Figure 5: Three marker objects; the second has its internal menuform displayed. The marker object has a variety of uses in application programs.

G.1. xvw_create_marker() — *create a marker object*

Synopsis

```
xvobject xvw_create_marker(
   xvobject parent,
   char   *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the marker object (for use in app-defaults files, etc)

Returns

The marker object on success, NULL on failure

Description

A marker visual object supports the display of a marker; A wide variety of marker types are supported.

The (x, y) location of the marker center may be specified in world coordinates, where the world coordinate range is dictated by the "controlling" visual object. By default, the parent of the marker object is its controlling visual object; this may be changed using the XVW_GRAPHICS_ATTACH attribute.

Alternatively, the location of the marker may be specified using device coordinates. When device coordinates are used, they are with respect to the bounding box surrounding the marker; specify the (x,y) location of the upper left hand corner of the bounding box, as well as the width and height of the bounding box.

Because the size of a marker is automatically determined according to the scale, width and height should not be specified; instead, to increase the size of the marker, provide the desired scale value as an integer multiple of the original (default) size.

G.2. Attributes of the Marker Visual Object

Summary of Marker Attributes

Attribute	Description
XVW_GRAPHICS_MARKERTYPE	The marker type.
XVW_MARKER_XPLACEMENT	Specifies the x location of the center of the marker in world coordinates.
XVW_MARKER_YPLACEMENT	Specifies the y location of the center of the marker in world coordinates.

Descriptions of Marker Attributes

Attribute (Resource Name)	Type	Default	Legal Values

Descriptions of Marker Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_GRAPHICS_MARKERTYPE (graphicsMarkertype)	int	KMARKER_SQUARE	KMARKER_NONE KMARKER_ARC KMARKER_BOW_TIE KMARKER_BOX KMARKER_CARET KMARKER_CIRCLE KMARKER_CROSS KMARKER_DAGGER KMARKER_DIAMOND KMARKER_DOT KMARKER_HEXAGON KMARKER_POINT KMARKER_SQUARE KMARKER_TRIANGLE KMARKER_X KMARKER_V KMARKER_PIXEL
XVW_MARKER_XPLACEMENT (N/A)	double	0.0	any double world coordinate value
XVW_MARKER_YPLACEMENT (N/A)	double	0.0	any double world coordinate value

G.3. Complete Resource Set of the Marker Visual Object

The inheritance tree of the marker object is as follows:

manager -> graphics -> marker

Accordingly, the complete resource set for the marker object includes:

1. The marker attribute resource set, given above
2. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

G.4. Example using the Marker Visual Object

An example program using the marker visual object can be found in `$ENVISION/examples/annotate/01.marker/`. This program is as follows.

```
#include <envision.h>

/*
 * This program demonstrates the use of the marker visual object;
 * it creates a manager object containing several markers using
 * different foreground colors.
 *
 * The marker object may be selected and moved by holding down the shift key and
 * using the left mouse button. The menuform for the marker annotation may be
 * brought up by holding down the meta key and clicking the middle mouse button;
 * the menuform may be used to change attributes of the marker such as type,
 * scale, color, and world coordinate placement.
 */

void main(int  argc,
   char *argv[])
{
     int    i;
        xvobject marker;
        xvobject parent;
     static char *list[] = {"yellow", "orange", "red", "violet",
                     "purple", "blue", "green", "yellowgreen"};

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

     /* initialize the xvwidgets library */
        if (!xvw_initialize(XVW_MENUS_XVFORMS))
        {
           kerror(NULL, "main", "unable to open display");
           kexit(KEXIT_FAILURE);
        }

     /*
      * create a manager object to be the parent of all the markers;
          * make the background black
          */
     parent = xvw_create_manager(NULL, "parent");
     xvw_set_attributes(parent,
                  XVW_BACKGROUND_COLOR, "black",
                  XVW_WIDTH,            450,
                  XVW_HEIGHT,           450,
                  NULL);

     /*
      * create one marker for each color in the list;
      * set its placement, color, type, and scale
      */
     for (i = 0; i < knumber(list); i++)
     {
            marker = xvw_create_marker(parent, "marker");
            xvw_set_attributes(marker,
                               XVW_MARKER_XPLACEMENT,   (i+1) * 0.1,
                               XVW_MARKER_YPLACEMENT,   (i+1) * 0.1,
```

```
                                XVW_FOREGROUND_COLOR,     list[i],
                                XVW_GRAPHICS_MARKERTYPE,  KMARKER_SQUARE,
                                XVW_GRAPHICS_MARKERSCALE, 2,
                                NULL);
        }

        /* display & run the program */
           xvf_run_form();
}
```

H. The Polyline Object

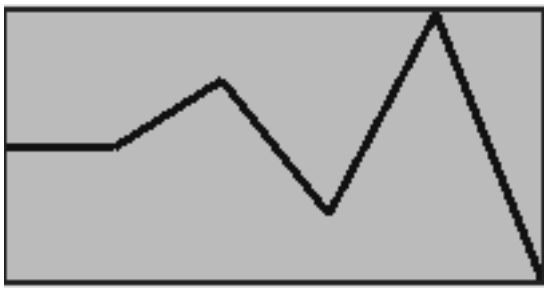


Figure 6: A polyline object.

H.1. xvw_create_polyline() — *create a polyline object.*

Synopsis

```
xvobject xvw_create_polyline(
   xvobject parent,
   char   *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the polyline object (for use in app-defaults files, etc)

Returns

The polyline object on success, NULL on failure

Description

The polyline visual object supports the display of a polyline.

The (x, y) location of each vertex of the polygon must be specified in world coordinates, where the world coordinate range is dictated by the "controlling" visual object. By default, the parent of the circle object is its controlling visual object; this may be changed using the XVW_GRAPHICS_ATTACH attribute.

Device coordinate specification of polyline vertices is not supported, as specification of the bounding box is not sufficient to determine location of vertices in the polygon.

NOTE: the polyline object is still under construction; interactive movement, resizing, and display of menuform is not yet available.

H.2. Attributes of the Polyline Object

Summary of Polyline Attributes

Attribute	Description
XVW_POLYLINE_NUMPTS	This is the number of data points contained in the Coord array specified by the attribute XVW_POLYLINE_POINTS. Note that you *must* use this attribute to specify the number of vertices prior to specifying the vertices with XVW_POLYLINE_POINTS.

XVW_POLYLINE_POINTS	This is the array of coordinates defining the vertices of the polyline in world coordinates. It is an array of type Coord, where the Coord structure is defined as:
	
	`typedef struct`
	`{`
	`  Real x, y, z;`
	`  Index d;`
	`} Coord;`
	Note that the XVW_POLYLINE_NUMPTS attribute *must* be set to the number of points in the Coord array *prior* to setting the XVW_POLYLINE_POINTS attribute.
	
	The *z* and *d* values are both ignored.

Descriptions of Polyline Attributes

Attribute (Resource Name)	Type	Default	Legal Values
XVW_POLYLINE_NUMPTS (N/A)	int	0	values > 0

Descriptions of Polyline Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_POLYLINE_POINTS (N/A)	Coord *	NULL	array of Coords of size given by XVW_POLYLINE_NUMPTS

H.3. Complete Resource Set of the Polyline Object

The inheritance tree of the polyline object is as follows:

manager -> graphics -> polyline

Accordingly, the complete resource set for the polyline object includes:

1. The polyline object attribute, given above
2. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

I. The Rectangle Object

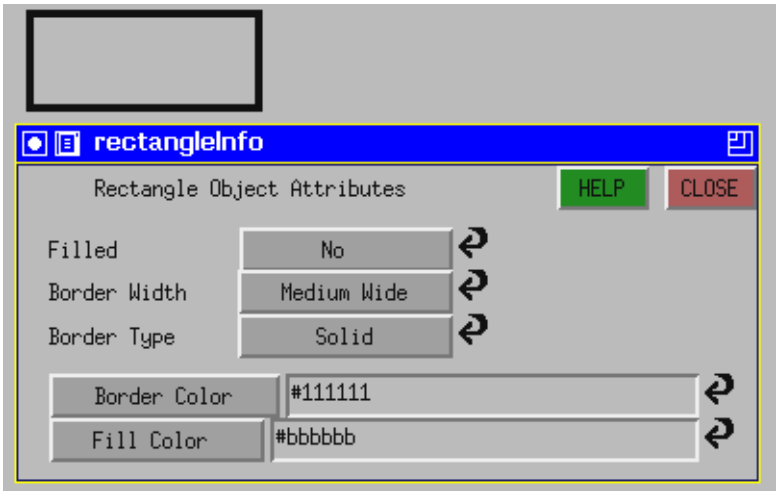


Figure 7: The rectangle object with its internal menuform displayed.

I.1. xvw_create_rectangle() — *creates a rectangle object*

Synopsis

```
xvobject xvw_create_rectangle(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the rectangle object (for use in app-defaults files, etc)

Returns

The rectangle object on success, NULL on failure

Description

A rectangle visual object supports the display of a rectangle.

The (x, y) location of the upper left hand corner of the rectangle may be specified in world coordinates, where the world coordinate range is dictated by the "controlling" visual object. By default, the parent of the rectangle object is its controlling visual object; this may be changed using the XVW_GRAPHICS_ATTACH attribute.

Alternatively, the size and location of the rectangle may be specified using device coordinates. As when using world coordinates, specify the (x,y) location of the upper left hand corner of the rectangle, as well as the width and height of the rectangle.

I.2. Attributes of the Rectangle Object

Summary of Rectangle Attributes

Attribute	Description
XVW_RECTANGLE_HEIGHT	This is the height of the rectangle in world coordinates.
XVW_RECTANGLE_WIDTH	This is the width of the rectangle in world coordinates.
XVW_RECTANGLE_X	This is the world coordinate location of the upper left hand
XVW_RECTANGLE_Y	This is the world coordinate location of the upper left hand corner of the rectangle.

Descriptions of Rectangle Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_RECTANGLE_HEIGHT (N/A)	double	0.0	value > 0.0
XVW_RECTANGLE_WIDTH (N/A)	double	0.0	value > 0.0
XVW_RECTANGLE_X (N/A)	double	0.0	any double value
XVW_RECTANGLE_Y (N/A)	double	0.0	any double value

I.3. Complete Resource Set of the Rectangle Object

The inheritance tree of the rectangle object is as follows:

manager -> graphics -> rectangle

Accordingly, the complete resource set for the rectangle object includes:

1. The rectangle attribute resource set, given above
2. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

I.4. Example Using the Rectangle Visual Object

An example of a programs using the Rectangle object can be found in `$ENVISION/examples/annotate/07.rectangle/`. This program is as follows.

```
#include <envision.h>

/*
 * This example creates a window with a single rectangle annotation in it.
 *
 * After the manager object parent is put into edit mode (hold down
 * shift key and click the left mouse button), the rectangle can be
 * interactively moved by "grabbing" it in the middle with the left mouse
 * button; grabbing it near an end will cause the rectangle to resize.
 *
 * In this example, the internal menuform for the rectangle is displayed
 * automatically.
 */

void main(
```

```
   int  argc,
   char *argv[])
{
        xvobject parent;
        xvobject rectangle;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

        /* initialize the xvwidgets lib */
        if (!xvw_initialize(XVW_MENUS_XVFORMS))
        {
           kerror(NULL, "main", "unable to open display");
           kexit(KEXIT_FAILURE);
        }

        /*
         *  create a black manager backplane, with width & height of 500.
         */
        parent = xvw_create_manager(NULL, "parent");
        xvw_set_attributes(parent,
                           XVW_WIDTH,            500,
                           XVW_HEIGHT,           500,
                           XVW_BACKGROUND_COLOR, "black",
                           NULL);
        /*
         *  create rectangle object, specifying dimensions in world
         *  coordinates, and foreground color of red.
         */
        rectangle = xvw_create_rectangle(parent, "rectangle");
        xvw_set_attributes(rectangle,
                     XVW_RECTANGLE_X,      0.5,
                     XVW_RECTANGLE_Y,      0.5,
                     XVW_RECTANGLE_WIDTH,  0.15,
                     XVW_RECTANGLE_HEIGHT, 0.05,
                     XVW_FOREGROUND_COLOR, "red",
                     XVW_GRAPHICS_FILLED,  "TRUE",
                     XVW_BACKGROUND_COLOR,  "purple",
                     NULL);
        /*
         *  activate menuform so user doesn't have to bring it up.
         */
        xvw_activate_menu(rectangle);

        /* display & run the program */
        xvf_run_form();
}
```

J. The String Object

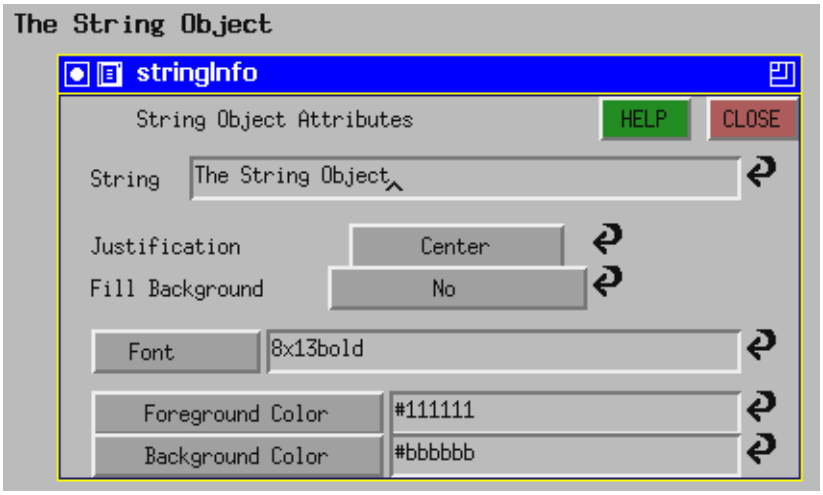


Figure 8: The string object, displaying string "The String Object", with its internal menuform displayed.

J.1. xvw_create_string() — *create a string annotation*

Synopsis

```
xvobject xvw_create_string(
    xvobject parent,
    char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the string object (for use in app-defaults files, etc)

Returns

The string object on success, NULL on failure

Description

A string visual object is used to display a (non-editable) string of characters. The string specified may be of any length, and of multiple lines. Carriage returns may be specified with "\n".

The location of the upper left corner of the the string may be specified in world coordinates, where the world coordinate range is dictated by the "controlling" visual object. By default, the parent of the string object is its controlling visual object; this may be changed using the XVW_GRAPHICS_ATTACH attribute.

Alternatively, the location of the string may be specified using device coordinates. As when using world coordinates, specify the upper left hand corner of the string.

The amount of space used by the string cannot be explicitly set by the application; this is automatically calculated according to the font that is used. Specification of a width and height simply provides a buffer of space around the string; note that justification will have no effect unless the width and height are specified to be larger than what is actually needed by the string. Width and height may be specified in characters or in pixels.

In addition to font specification, the string object also supports multiple styles in which the string may appear: available styles include plain, emphasized, embossed in, and embossed out.

J.2. Attributes of the String Object

Summary of String Attributes

Attribute	Description
XVW_STRING_HIGHLIGHT_COLOR	When the XVW_STRING_STYLE attribute is used to set the string style to KSTRING_STYLE_EMPHASIZE, KSTRING_STYLE_EMBOSSED_IN, or KSTRING_STYLE_EMBOSSED_OUT, this attribute determines the additional color to be used in the process of drawing the emphasized or embossed string.
XVW_STRING_HIGHLIGHT_PIXEL	The pixel value that defines the desired highlight color.
XVW_STRING_JUSTIFICATION	This attribute specifies the justification of the string *with respect to the width and height of the string object*. Thus, if the string object has its string set to "hello", with a width of 5 and a height of 1, the justification will have little (if any) effect, as the string has no room to "move" within its boundaries. However, if the width and height are set to *larger* than what is necessary to display the string, the justification will take effect. Justification can be set to any of the following values: KSTRING_JUSTIFY_CENTER ("Center" on the menuform) KSTRING_JUSTIFY_TOP ("Top" on the menuform) KSTRING_JUSTIFY_BOTTOM ("Bottom" on the menuform) KSTRING_JUSTIFY_LEFT ("Left" on the menuform) KSTRING_JUSTIFY_RIGHT ("Right" on the menuform) KSTRING_JUSTIFY_TOPRIGHT ("TopRight" on the menuform) KSTRING_JUSTIFY_TOPLEFT ("TopLeft" on the menuform) KSTRING_JUSTIFY_BOTTOMRIGHT ("BottomRight" on the menuform) KSTRING_JUSTIFY_BOTTOMLEFT ("BottomLeft" on the menuform)
XVW_STRING_STRING	This is the string to display in the string object.

Summary of String Attributes	
Attribute	**Description**
XVW_STRING_STYLE	The string visual object supports three styles: plain, emphasized, and embossed. When this attribute is set to KSTRING_STYLE_PLAIN, the string is drawn once, in the foreground color. When this attribute is set to KSTRING_STYLE_EMPHASIZE, the string is emphasized. This means it is double-drawn using both the foreground and highlight colors; this causes it to take on a 3D, or "emphasized" effect. When this attribute is set to KSTRING_STYLE_EMBOSSED_IN, the string will appear embossed "into" the background; both the foreground and highlight colors are used in embossing. KSTRING_STYLE_EMBOSSED_OUT also causes the string to appear embossed, but the string will seem to come "out of" rather than "into" the background.
XVW_STRING_XPLACEMENT	This double value specifies the x location of the lower left hand corner of the string in world coordinates.
XVW_STRING_YPLACEMENT	This double value specifies the y location of the lower left hand corner of the string in world coordinates.

Descriptions of String Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_STRING_HIGHLIGHT_COLOR (stringHighlightColor)	char *	default bg color	any valid color name: see /usr/lib/X11/rgb.txt or Section 7.1.1 of *The XLib Programming Manual* by Adrian Nye
XVW_STRING_HIGHLIGHT_PIXEL (N/A)	unsigned long	default bg pixel (XtDefaultBack-ground)	any valid pixel value: see Section 7.3 and 7.4 of *The XLib Programming Manual* by Adrian Nye
XVW_STRING_JUSTIFICATION (stringJustification)	int	KSTRING_JUSTIFY_LEFT	KSTRING_JUSTIFY_CENTER KSTRING_JUSTIFY_TOP KSTRING_JUSTIFY_BOTTOM KSTRING_JUSTIFY_LEFT KSTRING_JUSTIFY_RIGHT KSTRING_JUSTIFY_TOPRIGHT KSTRING_JUSTIFY_TOPLEFT KSTRING_JUSTIFY_BOTTOMRIGHT KSTRING_JUSTIFY_BOTTOMLEFT
XVW_STRING_STRING (N/A)	char *	NULL	any printable string
XVW_STRING_STYLE (stringEmphasize)	int	KSTRING_STYLE_PLAIN	KSTRING_STYLE_PLAIN KSTRING_STYLE_EMPHASIZE KSTRING_STYLE_EMBOSSED_IN KSTRING_STYLE_EMBOSSED_OUT
XVW_STRING_XPLACEMENT (N/A)	double	0.0	any double value

Descriptions of String Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_STRING_YPLACEMENT (N/A)	double	0.0	any double value

J.3. Attributes of the String Object

The inheritance tree of the string object is as follows:

manager -> graphics -> string

Accordingly, the complete resource set for the string object includes:

1. The string attribute resource set, given above
2. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

J.4. Example Using the String Visual Object

An examples of a program using the string object can be found in `$ENVISION/examples/annotate/08.string`. This program is as follows.

```
#include <envision.h>

/*
 *
 * This example creates a window with a single string annotation in it.
 *
 * After the manager object parent is put into edit mode (hold down
 * shift key and click the left mouse button), the string can be
 * interactively moved by "grabbing" it in the middle with the left mouse
 * button; grabbing it near an end will cause the string to resize.
 *
 * In this example, the internal menuform for the string is displayed
 * automatically.
 */

void main(
   int  argc,
   char *argv[])
{
        xvobject parent;
        xvobject string;
```

```
        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

        /* initialize the xvwidgets lib */
        if (!xvw_initialize(XVW_MENUS_XVFORMS))
        {
           kerror(NULL, "main", "unable to open display");
           kexit(KEXIT_FAILURE);
        }

        /*
         *  create a manager backplane, width & height of 300.
         */
        parent = xvw_create_manager(NULL, "parent");
        xvw_set_attributes(parent,
                           XVW_WIDTH,      300,
                           XVW_HEIGHT,     300,
                           NULL);

        /*
         *  create the string object, specifying placement in world coordinates.
         *  an overly large character width & height is given so that
      *  justification can be tested.  specify string to be displayed and
      *  style to display it in.
         */
        string = xvw_create_string(parent, "string");
        xvw_set_attributes(string,
                     XVW_STRING_XPLACEMENT,    0.5,
                     XVW_STRING_YPLACEMENT,    0.5,
                  XVW_CHAR_WIDTH,           25.0,
                  XVW_CHAR_HEIGHT,          10.0,
                     XVW_STRING_STRING,      "Test\n String",
                     XVW_STRING_JUSTIFICATION, KSTRING_JUSTIFY_CENTER,
                     XVW_STRING_STYLE,         KSTRING_STYLE_EMBOSSED_OUT,
                     NULL);
        /*
         *  activate the menuform so user doesn't have to bring it up.
         */
        xvw_activate_menu(string);

        /* display & run the program */
        xvf_run_form();
}
```

K. The StringValue Object

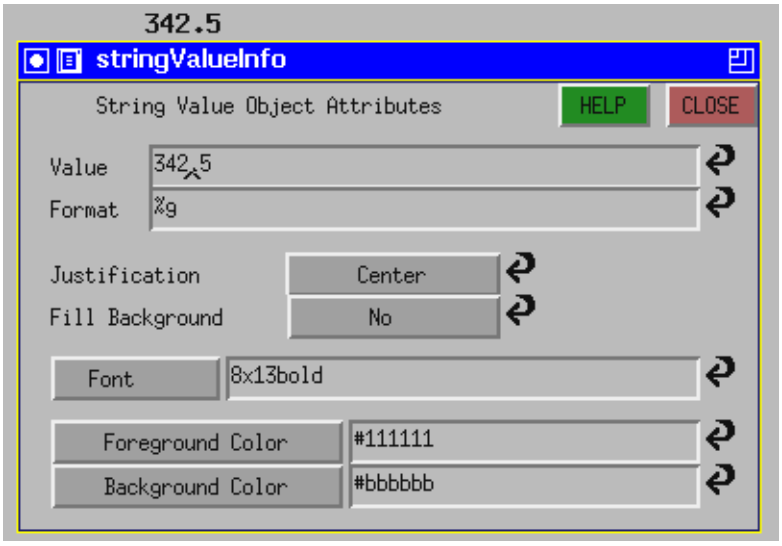


Figure 9: The stringvalue object with its internal menuform displayed. The stringvalue object is used to display a formatted number as a string.

K.1. xvw_create_stringvalue() — *creates a string value object*

Synopsis

```
xvobject xvw_create_stringvalue(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel be created automatically

`name`
: a name for this particular instance of the stringvalue object (for use in app-defaults files, etc)

Returns

The stringvalue object on success, NULL on failure

Description

A stringvalue object supports the display of a formatted number. Thus, standard C formatting strings such as "%g" or "%.3f" may be used to neatly display a floating point or double precision value.

The location of the upper left corner of the the stringvalue may be specified in world coordinates, where the world coordinate range is dictated by the "controlling" visual object. By default, the parent of the stringvalue object is its controlling visual object; this may be changed using the XVW_GRAPHICS_ATTACH attribute.

Alternatively, the location of the stringvalue may be specified using device coordinates. As when using world coordinates, specify the upper left hand corner of the stringvalue.

The amount of space used by the stringvalue cannot be explicitly set by the application; this is automatically calculated according to the font that is used. Specification of a width and height simply provides a buffer of space around the text; note that justification will have no effect unless the width and height are specified to be larger than what is actually needed by the text displaying the date. Width and height may be specified in characters or in pixels.

In addition to font specification, the stringvalue object also supports multiple styles in which the specified number can appear: available styles include plain, emphasized, embossed in, and embossed out.

K.2. Attributes of the StringValue Object

Summary of StringValue Attributes	
Attribute	**Description**
XVW_STRING_FORMAT	This is the format in which the string object is to display the numerical value. Legal values are identical to those used with *printf()* and *scanf()*, except for %s and %c.
XVW_STRING_VALUE	This is the numerical value to display.

Descriptions of StringValue Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_STRING_FORMAT (stringFormat)	char *	"%g"	see description
XVW_STRING_VALUE (N/A)	double	KMAXFLOAT	any double value

K.3. Complete Resource Set of the StringValue Object

The inheritance tree of the stringvalue object is as follows:

manager -> graphics -> string -> stringvalue

Accordingly, the complete resource set for the stringvalue object includes:

1. The string value attribute resource set, given above

2. The string attribute resource set, given in the previous section

3. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"

4. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

K.4. Example Using the StringValue Visual Object

An example of a program using the stringvalue object can be found in `$ENVISION/examples/annotate/09.stringvalue`. This program is as follows

```
#include <envision.h>

/*
 *
 * This example creates a window with a stringvalue annotation; the stringvalue
 * visual object is used to display a floating point number in a particular
 * format.
 *
 * After the manager object parent is put into edit mode (hold down
 * shift key and click the left mouse button), the stringvalue object can be
 * interactively moved by "grabbing" it in the middle with the left mouse
 * button; grabbing it near an end will cause the stringvalue object to resize.
 *
 * In this example, the internal menuform for the stringvalue object
 * is displayed automatically.
 */

void main(
   int  argc,
   char *argv[])
{
        xvobject parent;
        xvobject stringvalue;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "ENVISION");

        /* initialize the xvwidgets lib */
        if (!xvw_initialize(XVW_MENUS_XVFORMS))
        {
           kerror(NULL, "main", "unable to open display");
           kexit(KEXIT_FAILURE);
        }

        /*
         *  create a manager backplane, width & height of 300
         */
        parent = xvw_create_manager(NULL, "parent");
        xvw_set_attributes(parent,
                           XVW_WIDTH,      300,
                           XVW_HEIGHT,     300,
                           NULL);
```

```
        /*
         *  create stringvalue object, specifying placement in world
       *  coordinates, give justification, and set value to be displayed
         */
        stringvalue = xvw_create_stringvalue(parent, "stringvalue");
        xvw_set_attributes(stringvalue,
          XVW_STRING_VALUE,    10.2,
          XVW_STRING_XPLACEMENT,     0.5,
          XVW_STRING_YPLACEMENT,     0.5,
          XVW_STRING_JUSTIFICATION, KSTRING_JUSTIFY_CENTER,
          NULL);

        /*
         *  activate the menuform so user doesn't *  have to bring it up.
         */
        xvw_activate_menu(stringvalue);

        /* display & run the program */
        xvf_run_form();
}
```

L. The Timer Object

Figure 10: The timer object acts like a stopwatch, starting at the time specified and using the system clock to update as often as specified.

L.1. xvw_create_timer() — *create a timer object.*

Synopsis

```
xvobject xvw_create_timer(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: a name for this particular instance of the timer object (for use in app-defaults files, etc)

Returns

The timer object on success, NULL on failure

Description

The timer object was designed for use as a stopwatch. It displays the current time (including seconds) according to the system clock of the computer. It can be started and stopped as desired, and the update time can be specified to fractions of a second.

The location of the upper left corner of the timer may be specified in world coordinates, where the world coordinate range is dictated by the "controlling" visual object. By default, the parent of the timer object is its controlling visual object; this may be changed using the XVW_GRAPHICS_ATTACH attribute.

Alternatively, the location of the timer may be specified using device coordinates. As when using world coordinates, specify the upper left hand corner of the timer.

The amount of space used by the text displaying the time cannot be explicitly set by the application; this is automatically calculated according to the font that is used. Specification of a width and height simply provides a buffer of space around the time; note that justification will have no effect unless the width and height are specified to be larger than what is actually needed by the text displaying the time. Width and height may be specified in characters or in pixels.

In addition to font specification, the timer object also supports multiple styles in which the string displaying the time can appear: available styles include plain, emphasized, embossed in, and embossed out.

L.2. Attributes of the Timer Object

Summary of Timer Attributes

Attribute	Description
XVW_TIMER_COUNTER	This attribute specifies the time (in seconds) at which the timer is started. Usually, this value is left at the default, 0.0, as (like a stopwatch) the timer is considered to be "turned on" at time = 0.0. However, there may are occasions when the "turn on" time is at some time later than 0.
XVW_TIMER_UPDATETIME	This attribute indicates how often the timer is updated, given in fractions of a second. If, for example, XVW_TIMER_UPDATETIME is set to 0.25, this indicates that the timer will update every 0.25 seconds, or 4 times a second. Note that the timer can only be as accurate as your computer system clock.

Descriptions of Timer Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_TIMER_COUNTER (N/A)	double	0.0	values >= 0.0
XVW_TIMER_UPDATETIME (timerUpdatetime)	double	0.1	values > 0.0

L.3. Attributes of the Timer Object

The inheritance tree of the timer object is as follows:

manager -> graphics -> string -> stringvalue -> timer

Accordingly, the complete resource set for the timer object includes:

1. The timer object attribute, given above
2. The graphics attribute resource set, given in Chapter 5, "The Graphics Attributes"
3. The general object attributes, given in Chapter 2, "The xvwidgets Library," Section B, "General Attributes of GUI and Visual Objects."

This page left intentionally blank

Table of Contents

Chapter 8

Xvforms

Chapter 8 - Xvforms

A. Introduction

Most VisiQuest 2001 software objects [1] have a Graphical User Interface, or GUI which is defined by a User Interface Specification File, or UIS file. The GUI of a software object may be displayed for a variety of reasons:

1. The user may be in the process of modifying or examining the GUI, with the use of the guise GUI specification editor, or with the preview GUI display tool.

2. The user may be using the VisiQuest visual language to access the software object, and has displayed the GUI for the program by clicking on the "menu" icon in the middle of the glyph which represents the program.

3. The user has used the xvrun tool to display the GUI of the software object, and will execute the process via the GUI.

4. The user has used the built-in `[-gui]` option of the program to display its GUI, and is using the GUI of the program as an alternative to providing program arguments on the command line.

5. If the software object is an xvroutine, by its nature it must display its GUI when it is executed from the command line.

6. If the software object is a library which implements GUI or visual objects (such as the *xvisual* library), the user can put a displayed object in *edit mode*, and bring up its *internal menuform* GUI that offers the user control over its attributes.

The *xvforms* library creates, displays, and maintains the GUI of a VisiQuest 2001 software object.

A.1. Available GUI Items

Assuming some prior experience with the VisiQuest 2001 system, most of the items that comprise a VisiQuest 2001 GUI are probably already familiar; however, this section provides a review of the GUI items supported by the *xvforms* library.

---

[1] including kroutines, xvroutines, pane objects, script objects, and those library objects which use *internal menuforms*

A.1.1. The Form

Item Description

Any VisiQuest GUI, as a whole, is referred to as a "form".

Use by the GUI

The form *is* the GUI.

Use by the CLUI

A form is not applicable to a CLUI.

A.1.2. The Subform

Item Description

Every GUI has at least one subform. For simple user interfaces, a single subform is all that is necessary. The subform provides a backplane for one or more panes. If the subform has more than one pane, it will also have a guide pane with which to change the pane that is currently displayed.

Use by the GUI

The subform serves as a backplane and may contain one or more *panes*.

Use by the CLUI

A subform is not relevant to the CLUI.

A.1.3. The Pane

Item Description

Every GUI has at least one pane. For very simple GUI's, a single pane on a single subform is all that is necessary. For more complex GUI's, a subform may contain many panes; the user switches between panes by clicking on their associated guide buttons.

Use by the GUI

The pane provides a backplane for GUI selections.

Use by the CLUI

The pane is not used by the CLUI directly.

A.1.4. The Master Form

Item Description

When more than one subform is needed on the GUI, a *master form* is necessary. A master form always contains one or more *subform buttons* which allow the user to control which subform is currently displayed; while the currently displayed subforms be mapped and unmapped, the master form remains displayed throughout the run of the program. A master form may also include action buttons and selections, but these should be limited to only those which would otherwise be repeated on each of the subforms in the GUI.

Use by the GUI

The master form provides a backplane for subform buttons and selections.

Use by the CLUI

The master form is not applicable to the CLUI.

A.1.5. The Guide Pane

Item Description

When more than one pane is needed on a subform, a *guidepane* is necessary. A guidepane always contains two or more *guide buttons* which allow the user to control which pane is currently displayed on the subform. A guidepane may also include subform action buttons and selections, but these should be limited to only those which would otherwise be repeated on each of the panes of the subform.

Use by the GUI

The guidepane provides a backplane for guide buttons and selections.

Use by the CLUI

The guidepane is not relevant to the CLUI.

A.1.6. Subform Buttons

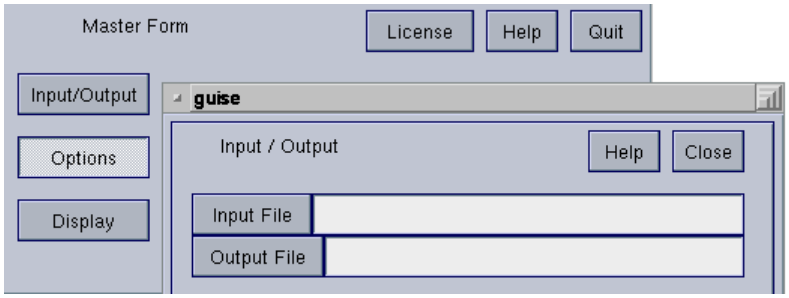


Figure 1: A subform button is used on a master form to map the subform with which it is associated.

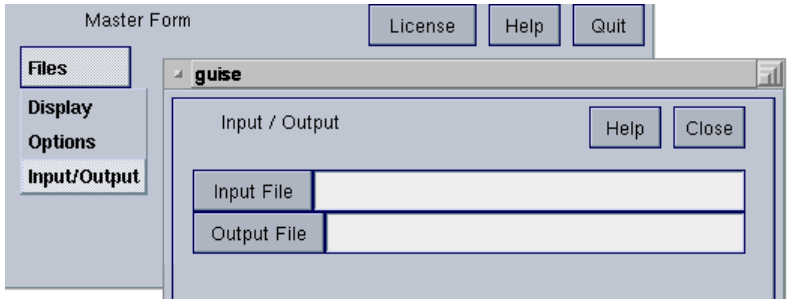


Figure 2: Subform buttons and guide buttons are frequently gathered into a pulldown menu.

Item Description

When multiple subforms are used, it is necessary to have a mechanism on the master form with which to change the subform that is currently displayed (if subforms are mutually exclusive) or to bring up new subforms (is subforms are not mutually exclusive). The subform button, situated on the master form, provides this mechanism. Each subform has its own subform button; when the user clicks on the subform button, that subform is displayed. Each subform button is associated with its subform with the use of the *variable*, which must be identical for the subform button and the subform with which it is associated. Subform buttons are used exclusively in the *.form files for *xvroutines*.

Use by the GUI

The user clicks on a subform button in order to map the subform with which it is associated. Mapping of the subform is done automatically.

Use by the CLUI

The line in the UIS file representing the subform button is ignored by the CLUI.

A.1.7. Guide Buttons

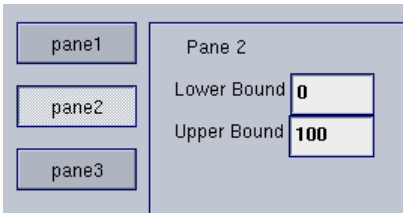


Figure 3: A guide button lets the user control which pane in a multi-pane subform is currently mapped. In this excerpt from a sample GUI, the three guide buttons in the guidepane to the left are used to switch among the three panes on the right. Currently, the guide button "Pane2" is used to display the second pane, named "Pane 2".

Item Description

When a subform contains more than one pane, it is necessary to have a mechanism with which to change the pane that is currently displayed. The guide button, situated on the guide pane, provides this mechanism. Each pane has its own guide button; when the user clicks on the guide button, that pane is displayed. Only one pane may be displayed in a subform at any given time. Each guide button is associated with its pane with the use of the *variable*, which must be identical for the guide button and the pane with which it is associated. Guide buttons are used exclusively in the *.form file for *xvroutines*.

Use by the GUI

The user clicks on a guide button in order to map the pane with which it is associated. Mapping of the pane is done automatically. Only one pane may be mapped at any one time.

Use by the CLUI

The line in the UIS file representing the guide button is ignored by the CLUI.

A.1.8. Action Buttons

Figure 4: A pane action lets the user request a particular action; in this case, the action will be to "edit".

Item Description

The purpose of an action button is to return software control to the application program; it is used exclusively in the *.form file by *xvroutines*. In general, when a pane contains two or more non-

live selections, an action button should be placed at the bottom of the pane so that the user can tell the application, "I've finished setting values of selections now, go ahead and perform (some action)." Alternatively, an application may be able to perform a particular operation without any additional information provided by the user besides the fact that the user now wants the operation performed; this is the other case in which an action button is appropriate. A mouse click on the action button causes software control to be diverted from the graphical user interface to the application's subroutine that is associated with the action button, where the name of the subroutine in question is determined by the variable associated with the action button, as well as the variable associated with the pane, guide pane, or master form on which the action button appears.

Use by the GUI

The user clicks on an action button to request a particular action from the application. Control is immediately returned from the GUI to the application so that the action may be performed.

Use by the CLUI

The line in the UIS file representing the action button is ignored by the CLUI.

A.1.9. Help Buttons

Figure 5: Every master form, subform, and pane should have one help button that allows the user to access an online help page that covers that part of the GUI.

Item Description

The help button is a standard device on the GUI; all programs in the VisiQuest system are required to provide online help that can be accessed directly via the user interface. The help button provides access to a help file involving no additional work by the application program; all that is needed is the correct path to the help file. VisiQuest standards specify that help buttons always provide help pages specific to their location. Thus, on a GUI with all three levels of hierarchy, the help button located on the master form will give information about the program as a whole, the help button located on a subform will give information about the general options offered by that subform, and the help button located on a pane will give detailed information about the I/O located on that pane. One help button should be used for each pane, each subform, and the master form (if any) in the *.form file for *xvroutines*. Every *.pane file should include one help button to access the man page for the program.

Use by the GUI

The help button provides access to online help from the GUI.

Use by the CLUI

The UIS line representing the help button is ignored by the CLUI.

A.1.10. Quit Buttons

Figure 6: A quit button allows the user to quit the program. Quit buttons are also used to close panes and subforms.

Item Description

The quit button allows the program to exit. In addition, when a GUI has a master form with several subforms, quit buttons on the subforms allow the user to unmap them and thus reduce clutter on the screen. When used to exit the program, the quit button is by convention labeled "Exit;" when used to close a subform, the quit button is by convention labeled "Close."

Use by the GUI

Allows the user to close a subform or exit the program.

Use by the CLUI

The UIS line specifying the Quit button is ignored by the CLUI.

A.1.11. InputFile Selections

Figure 7: An Input File selection; the title is on a button that brings up the VisiQuest file/aliases browser.

Item Description

This allows the user to specify an input file; the file will be checked for existence and read permission. An input file may be part of a toggle, or a member of a mutually inclusive or mutually exclusive group. It may be optional or required. It may be used in the *.form file for an *xvroutine*, or in the *.pane file for any VisiQuest program.

Use by the GUI

When used by the GUI, the *input file* selection allows the user to enter an input file directly or to use the file browser to select a file. The input file selection consists of an optional box (if the selection is optional), a title, and a text box in which the user may enter a filename. The title of the input file selection is actually a button that can be used to bring up the file browser.

Use by the CLUI

When used by the CLUI, the input file specifies an *input file argument*, as in "-i1 my_input_image.viff". If an invalid input file argument is entered, the user will be re-prompted. When an optional input file argument is absent from the command line, it takes on the default value specified (the default value may be NULL).

A.1.12. OutputFile Selections

Figure 8: The title of an Output File selection is a button that brings up the VisiQuest file/aliases browser.

Item Description

The OutputFile allows the user to specify an output file that will be checked for write permission. OutputFile lines may be part of a toggle, or members of a mutually inclusive or mutually exclusive group. They may be optional or required. It may be used in the *.form file for an *xvroutine*, or in the *.pane file for any VisiQuest program.

Use by the GUI

When used by the GUI, the *output file selection* allows the user to enter an output file directly or to use the file browser to select an existing output file. The output file selection consists of an optional box (if the selection is optional), a title, and a text box in which the user may enter a filename. The title of the output file selection is actually a button that can be used to bring up the file browser.

Use by the CLUI

When used by the CLUI, the output file specifies an *output file argument*, as in "-o1 my_output_file". The user will be re-prompted for invalid output file arguments. When an optional output file argument is absent from the command line, it takes on the default value specified (the default value may be NULL).

A.1.13. Integer Selections

Figure 9: An required Integer selection using a scrollbar.

Item Description

Integers may be part of a toggle, or members of a mutually inclusive or mutually exclusive group. They may be optional or required. Bounded or unbounded values may be specified. It may be used in the *.form file for an *xvroutine*, or in the *.pane file for any VisiQuest program.

Use by the GUI

When used by the GUI, the *integer selection* consists of an optional box (if the integer is optional) a title, and a text box in which the integer may be entered. Bounded integers may use an optional scroll bar. Live integers will have the stylized carriage return symbol appended to the right side.

Use by the CLUI

When used by the CLUI, the integer specifies an *integer argument*, as in "-int_variable 21". Values entered for bounded integer arguments will be checked to be sure they are within bounds; the user will be re-prompted for invalid entries. When an optional integer argument is absent from the command line, it takes on the default value specified.

A.1.14. Float Selections

Figure 10: An optional Float selection using a scrollbar.

Item Description

A float is used when a floating point number is needed. A float may be part of a toggle, or a member of a mutually inclusive or mutually exclusive group. It may be optional or required. Bounded or unbounded values may be specified. It may be used in the *.form file for an *xvroutine*, or in the *.pane file for any VisiQuest program.

Use by the GUI

When used by the GUI, the float specifies a *float selection*. The float selection consists of an optional box (if the float is optional), a title, and a text box in which the float may be entered. A bounded float may use an optional scroll bar. A live float will have the stylized carriage return symbol appended to the right side.

Use by the CLUI

When used by the CLUI, the float specifies a *float argument*, as in "-float_variable 0.123". Values entered for bounded float arguments will be checked to be sure they are within bounds; the user will be re-prompted for invalid and bogus entries. When an optional float argument is absent from the command line, it takes on the default value specified.

A.1.15. Double Selections

Figure 11: An optional Double selection.

Item Description

A double is used when double precision accuracy is needed. It may be part of a toggle, or a member of a mutually inclusive or mutually exclusive group. It may be optional or required. Bounded or unbounded values may be specified. It may be used in the *.form file for an *xvroutine*, or in the *.pane file for any VisiQuest program.

Use by the GUI

When used by the GUI, the double specifies a *double selection*. The double selection consists of an optional box (if the double is optional), a title, and a text box in which the double may be entered. A bounded double may use an optional scroll bar. A live double will have the stylized carriage return symbol appended to the right side.

Use by the CLUI

When used by the CLUI, the double specifies a *double argument*, as in "-dbl_variable 0.987654321". Values entered for a bounded double argument will be checked to be sure they are within bounds; the user will be re-prompted for out-of-bounds and bogus entries. When an optional double selection is absent from the command line, it takes on the default value specified.

A.1.16. String Selections

Figure 12: A string selection that will be used to enter a title.

Item Description

Strings may be part of a toggle, or members of a mutually inclusive or mutually exclusive group. They may be optional or required. It may be used in the *.form file for an *xvroutine*, or in the *.pane file for any VisiQuest program.

Use by the GUI

When used by the GUI, the *string selection* consists of an optional box (if the string is optional), a

title, and a text box in which the string may be entered. Live strings will have the stylized carriage return symbol appended to the right side.

Use by the CLUI

When used by the CLUI, the String specifies a *string argument*, as in "-string_variable 'my special string'". When an optional string selection is absent from the command line, it takes on the default value specified (the default value may be NULL). Any printable string is considered to be valid input.

A.1.17. Flag Selections

Figure 13: A flag selection allows the user to indicate whether or not to use a particular option, in this case, "Invert".

Item Description

A flag allows the input of an implied boolean value. A flag may be part of a toggle, or a member of a mutually inclusive or mutually exclusive group. It may be used in the *.form file for an *xvroutine*, or in the *.pane file for any VisiQuest program.

Use by the GUI

When used by the GUI, the flag specifies a *flag selection*. The flag selection consists of an optional box and a title. When the optional box is highlighted, the value of the flag is considered to be TRUE (1). When the optional box is un-highlighted, the value of the flag is considered to be FALSE (0). A live flag selection has the stylized carriage return symbol appended to the right side.

Use by the CLUI

When used by the CLUI, the flag specifies a *flag argument*. If the flag is provided on the command line, as in "-flag_variable," the argument is considered to be TRUE (1). If the flag is absent from the command line, the argument is considered to be FALSE (0).

A.1.18. Logical Selections

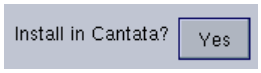


Figure 14: A logical selection that lets the user specify whether or not a software object should be installed in Cantata.

Item Description

A logical allows the input of an explicit boolean value. Logicals may be part of a toggle, or members of a mutually inclusive or mutually exclusive group. It may be used in the *.form file for an *xvroutine*, or in the *.pane file for any VisiQuest program.

Use by the GUI

When used by the GUI, the *logical selection* consists of an optional box (if the selection is optional), a title, and a button which may be switched back and forth between the two possible values. Live logical selections have the stylized carriage return symbol appended to the right side.

Use by the CLUI

When used by the CLUI, the Logical specifies a *logical argument*. The logical value is provided on the command line explicitly, as in "-logical 0" for a value of FALSE, or "-logical 1" for a value of TRUE. When an optional logical argument is absent from the command line, it takes on the default value specified. The user will be re-prompted for invalid entries.

A.1.19. Cycle Selections

Figure 15: A Cycle selection allowing the user to set the colorspace model, with three settings, "RGB", "CMY" and "HLS"; the current value is "RGB".

Item Description

A cycle allows the input of one of a number of predefined choices. Each choice is represented by both a number and a label, where the numbers are incremental integers beginning with an integer that you can specify. For example, you might have a cycle that moved through the sequence, "2 (dot), 3 (dash), 4 (dot-dash)". A Cycle may NOT be part of a toggle, but it may be a member of a mutually inclusive or mutually exclusive group. It may be optional or required. It may be used in the *.form file for an *xvroutine*, or in the *.pane file for any VisiQuest program.

Use by the GUI

When used by the GUI, the Cycle specifies a *cycle selection*. The cycle selection consists of an optional box (if the selection is optional), a title, and a button which may be cycled through the set of possible values. Live cycle selections have the stylized carriage return symbol appended to the right side. Because of their presentation, cycles are only recommended in situations where there are a small number (3 - 6) choices.

Use by the CLUI

When used by the CLUI, the Cycle specifies a *cycle argument*. The cycle value is provided on the command line explicitly, where either the integer or the label of a choice may be specified. In the above example, if the user wanted "dot", they might specify either "-cycle_variable 2" or "-cycle_variable dot". When an optional cycle argument is absent from the command line, it takes on the default value specified. The user will be re-prompted for invalid entries. Both the integer value and the label of the cycle value specified will be made available to the program.

A.1.20. List Selections

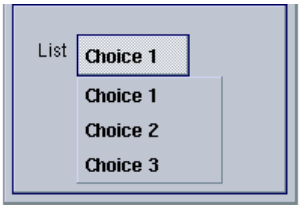


Figure 16: A list selection is used to select from three choices. A list selection differs from a displayed list selection in that it has a button with which to access the pulldown menu, rather than displaying the list all the time.

Item Description

A list allows the input of one of a number of predefined choices. Each choice is represented by both a number and a label, where the numbers are incremental integers beginning with an integer that may be specified. For example, you might have a list that offered the choices, "red (1), orange (2), yellow (3), green (4), blue(5) indigo (6)". Lists may be NOT be part of a toggle, but they may be members of a mutually inclusive or mutually exclusive group. They may be optional or required. It may be used in the *.form file for an *xvroutine*, or in the *.pane file for any VisiQuest program.

Use by the GUI

When used by the GUI, the *list selection* consists of an optional box (if the selection is optional), a title, and a button which displays a pulldown menu containing the possible values of the list. Live list selections have the stylized carriage return symbol appended to the right side. Because of their presentation, lists are recommended in situations where there are a large number (more than 5) choices. The *list selection* differs from the *displayed list selection* in that the *list selection*

features a pulldown menu that is accessable via a button, while the *displayed list selection* features a list the contents of which are always displayed.

Use by the CLUI

When used by the CLUI, the list specifies a *list argument*. The list value is provided on the command line explicitly, where either the integer or the label of a choice may be specified. In the above example, if the user wanted "red", they might specify either "-list_variable 1" or "-list_variable red". When an optional list argument is absent from the command line, it takes on the default value specified. The user will be re-prompted for invalid entries. Both the integer value and the label of the list value specified will be made available to the program.

A.1.21. DisplayList Selections

Figure 17: A displayed list selection has its list displayed all the time, in contrast to a list selection, which has a button with which to access the pulldown menu.

Item Description

A list allows the input of one of a number of predefined choices. Each choice is represented by both a number and a label, where the numbers are incremental integers beginning with an integer that you can specify. For example, you might have a list that offered the choices, "red (1), orange (2), yellow (3), green (4), blue(5) indigo (6)". Lists may be NOT be part of a toggle, but they may be members of a mutually inclusive or mutually exclusive group. They may be optional or required. It may be used in the *.form file for an *xvroutine*, or in the *.pane file for any VisiQuest program.

Use by the GUI

When used by the GUI, the *displayed list selection* consists of an optional box (if the selection is optional), a title, and a list of items from which the user may select. The list may have a scrollbar depending on the geometry of the selection. Live displayed list selections have the stylized carriage return symbol appended to the right of the label. Because of their presentation, displayed lists are recommended in situations where there is a large number (more than 5) choices. Depending on context, the actual size of the displayed list selection might be limited and the scrollbar used rather than allowing the displayed list to grow large enough to display all entries. The *displayed list selection* features a list the contents of which are always displayed, while the *list selection* features a pulldown menu which is accessable via a button.

Use by the CLUI

When used by the CLUI, the displayed list specifies a *list argument*. The list value is provided on the command line explicitly, where either the integer or the label of a choice may be specified. In the above example, if the user wanted "red", they might specify either "-list_variable 1" or "-list_variable red". When an optional list argument is absent from the command line, it takes on the default value specified. The user will be re-prompted for invalid entries. Both the integer value and the label of the list value specified will be made available to the program.

A.1.22. StringList Selections

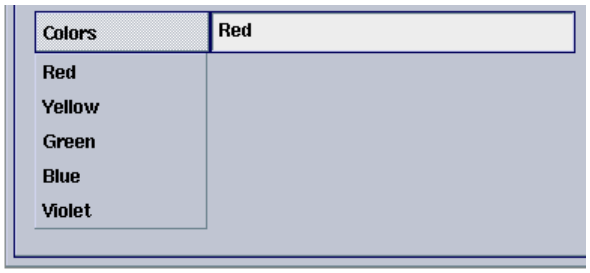


Figure 18: A stringlist selection offering the user one of a set of predefined colors, or the option to enter their own string.

Item Description

A stringlist allows the selection of predefined strings, or typing of a new string if the desired string is not in the predefined list. For example, you might have a "color" parameter where you know that the basic ROYGBIV colors are used most often, but you don't want to rule out the option of a less frequently used color. Stringlists may be NOT be part of a toggle, but they may be members of a mutually inclusive or mutually exclusive group. They may be optional or required. StringLists are used in the *.form file for *xvroutines*, and in the *.pane file for any VisiQuest program.

Use by the GUI

When used by the GUI, the *stringlist selection* consists of an optional box (if the selection is optional), a title, and a text box in which a string may be entered. The title is a button that will display a pulldown menu with the predefined values of the list. Live stringlist selections have the stylized carriage return symbol on the right side.

Use by the CLUI

When used by the CLUI, the StringList specifies a *stringlist argument*. The stringlist argument is treated identically to the string argument; the benefit of using a stringlist only applies to GUIs.

A.1.23. Blank Selections (Labels)

Figure 19: A blank selection is used to display text on the GUI.

Item Description

The Blank (label) provides a non-operational label widget on the GUI which may be used for general information and text display purposes. It may be used in the *.form file for an *xvroutine*, or in the *.pane file for any VisiQuest program.

Use by the GUI

The blank selection is used to display a label on the GUI

Use by the CLUI

The line in the UIS file representing the blank selection is ignored by the CLUI.

A.1.24. Routine Buttons

Figure 20: The routine button is used in *.pane files as part of the cantata GUI for a VisiQuest program. Commonly labelled, "Run", or "Execute", the routine button executes the program in question, using the values of the other selections on the pane to determine the arguments to pass the program.

Item Description

The Routine Button is used in the *.pane UIS file for all VisiQuest programs so that they may be integrated into the VisiQuest visual language. Routine buttons are used exclusively in the *.pane files of VisiQuest programs.

Use by the GUI

When the user clicks on this button, the specified program is immediately executed with the arguments specified by the values of all other selections on the pane.

Use by the CLUI

The line in the UIS file representing the routine button is ignored by the CLUI.

A.1.25. Stdin And Stdout Selections

Figure 21: A stdin selection is used only on pane objects that are being used to integrate non-VisiQuest programs (that are dependent on stdin for input) into the VisiQuest system.

Figure 22: A stdout selection is used only on pane objects that are being used to integrate non-VisiQuest programs (that are dependant on stdout for output) into the VisiQuest system.

Item Description

VisiQuest can be used as an *integration system*. This is appropriate when there exists a number of non-VisiQuest programs upon which one wishes to enforce standardized documentation, a consistent user interface, and accessability from the VisiQuest visual language, VisiQuest. The procedure that is followed when one is doing such an integration is to create a *pane object* for each program that is to be integrated; a pane object provides a VisiQuest GUI for each non-VisiQuest program. The stdin and stdout selections provide a mechanism whereby non-VisiQuest programs depending on stdin and stdout may be integrated into VisiQuest.

It is important to understand that programs written under the VisiQuest development system do not depend on stdin or stdout; instead, they have a formalized command line user interface where input and output files are specified using input file and output file arguments, as in "-i image:ball" or "-o my_image.viff".

However, if the programs to be integrated into VisiQuest depend on stdin and stdout for input and output, then some mechanism must be provided from within VisiQuest to accommodate them. The stdin and stdout selections were created for this reason.

Use by the GUI

The stdin and stdout selections are only used in the *.pane files for *pane objects* that are created in order to integrate non-VisiQuest, stdin- and stdout-dependent programs into VisiQuest.

Use by the CLUI

The lines in the UIS file representing stdin and stdout are ignored by the CLUI.

A.1.26. Submenus

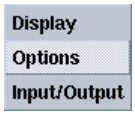


Figure 23: A submenu is created from button selections and blank selections.

Item Description

When many items are accessable from a particular master form, guidepane, or pane of the GUI, one can reduce clutter by grouping buttons and labels together onto a pulldown menu. Any group of selections that are made up of a *single* button or label may be collected into a submenu; thus, candidates for menu contents include: subform buttons (for submenus on master forms only) and guide buttons (for submenus on guide panes only) as well as action buttons, quit buttons, help buttons, and blank selections on any part of the GUI. It is most often used in the *.form file for an *xvroutine*. It may also be used in the *.pane file for any VisiQuest program, although by convention it is used sparingly in this context.

Use by the GUI

The submenu button presents a single button on the GUI; clicking on the button will display a pulldown menu from which any of the other buttons may be selected. Other buttons that are grouped into a submenu will retain their original function.

Use by the CLUI

The line in the UIS file representing the submenu is ignored by the CLUI.

A.1.27. Workspaces

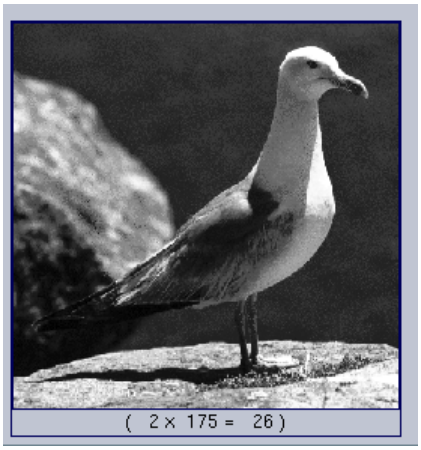


Figure 24: A workspace selection provides a general-purpose area on the GUI for whatever purpose may be appropriate for the application. Here, a workspace selection is used by editimage to hold the displayed image.

Item Description

The Workspace provides a general purpose manager widget on the GUI which may be used as a backplane for display of images, graphics, or special-purpose GUI elements that are not provided directly. The workspace is used exclusively in the *.form files of *xvroutines*.

Use by the GUI

The application may use the workspace for whatever purpose may be applicable. It has absolutely no functionality on its own; the application is provided with the address of the requested widget, and is then responsible for the use and maintenance of this widget. Once the workspace widget is displayed, the GUI does not manage or interfere with it further.

Use by the CLUI

The line in the UIS file representing the workspace is ignored by the CLUI.

B. About Public *xvforms* Library Calls

As mentioned earlier, the *xvforms* library creates, displays, and maintains the GUI of a VisiQuest 2001 software object. Kroutines, pane objects, and script objects are limited in that they may only have their GUI's displayed by another program, that program being an xvroutine such as preview, guise, VisiQuest, or xvrun. *Only xvroutines can make direct calls to the xvforms library to create, display, or modify their GUI's.*

Routines available in the *xvforms* library include routines to create a graphical user interface, run a graphical user interface, and change a graphical user interface during application execution.

Calls to some of the *xvforms* routines *must* be made in the main driver of every xvroutine. The required calls to these routines are generated automatically.

The *xvforms* library also includes other routines for modifying, maintaining, and changing the operation of the GUI of an xvroutine; calls to these routines may be added as desired. Of special interest is *xvf_set_attribute()*, which allows you to change the GUI of your xvroutine during execution.

The internal data structure used by the *xvforms* library to represent a GUI is referred to as a *form tree*. The data type of a form tree is the *kform*. The *xvf_create_form()* routine allocates and returns a *kform* pointer, while many of the other routines will take that same *kform* pointer as their first parameter. It is not necessary to understand the details of the form tree; however, some brief diagrams may help to illustrate the general structure:

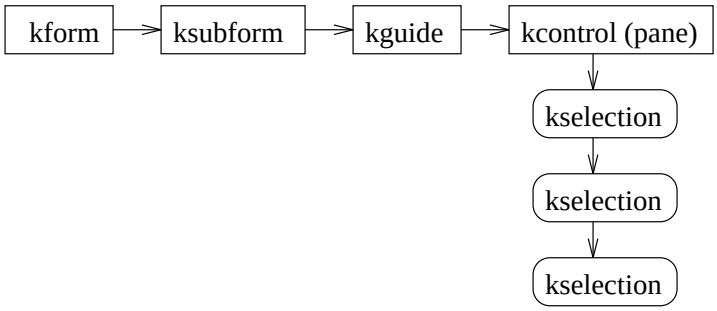


Figure 25: A simplified representation of the form tree associated with a GUI having a single pane on a single subform. Note that the data type associated with the form tree as a whole is the *kform*.

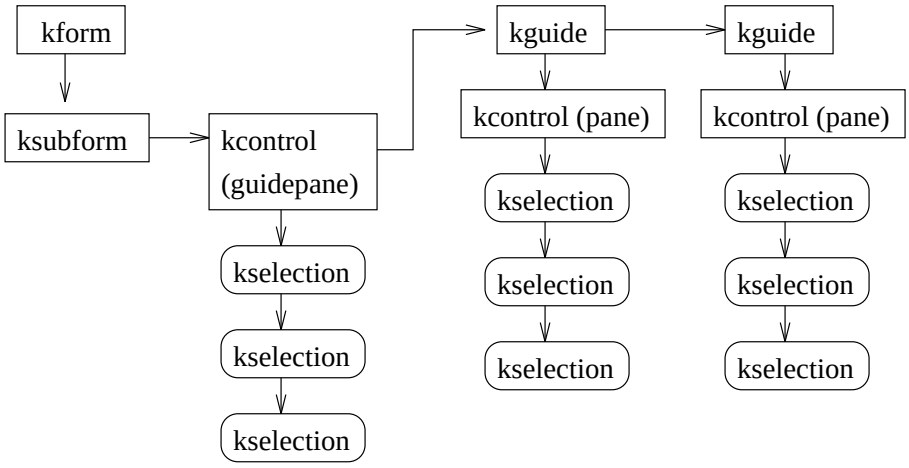


Figure 26: A simplified representation of the form tree associated with a GUI having a single subform with a guidepane and two panes. The guidepane has a variety of selections on it, as well as two guide buttons which provide access to the two separate panes.

The *kform* data type corresponds to the form tree as a whole, the *ksubform* data type corresponds to a subform branch of the form tree, the *kcontrol* data type corresponds to a pane branch of the form tree, the *kselection* data type corresponds to a GUI item leaf of the form tree, and so on.

Again, it is not important to understand the complexity of the form tree, except in a general way. An understanding of the form tree, however, does help to clarify the use of the *kform_struct* pointer. The xvroutine code generator will produce a pointer of data type *kform_struct* in the GUI information structure for each item on the GUI. This *kform_struct* pointer is generated specifically so that the xvroutine may call *xvf_set_attribute()* to modify its GUI during runtime if necessary.

A *kform_struct* is simply a "generic" form tree/branch pointer. That is, it may contain a pointer to a *kform*, a *ksubform*, a *kcontrol*, or a *kselection*; a type flag indicates which type of pointer it actually contains. Thus, calls to *xvf_set_attribute()* are the same, regardless of whether the portion of the form tree that will be modified is a subform branch, a pane branch, or a GUI item leaf.

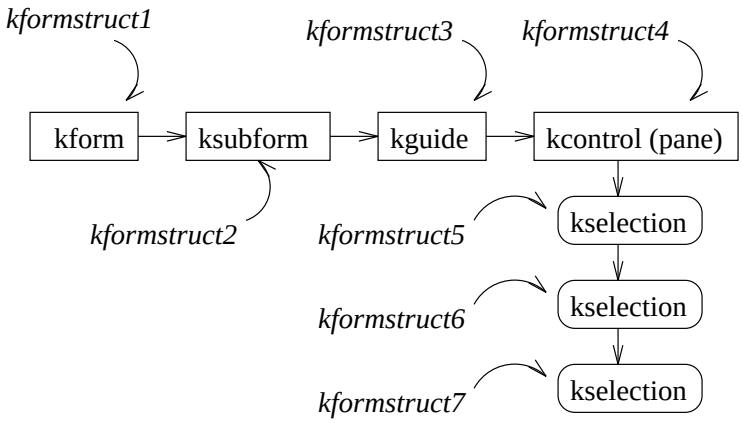


Figure 27: A *kform_struct* can be a pointer to any part of the form tree. Pointers of type *kform_struct* are automatically generated in the GUI Information structure for each selection on the GUI of an xvroutine. Then, these pointers may be used any time a call to *xvf_set_attribute()* is needed to change some attribute of a GUI item.

Having established the purpose of the *xvforms* library, the use of the *kform* pointer to the form tree and the *kform_struct* generic form tree/branch pointer, let's go on to examine the available routines. Public routines in the *xvforms* library include:

- *xvf_add_extra_call()* - add extra callback to GUI item
- *xvf_add_gui_callback()* - add callback to a GUI item
- *xvf_clear_selections()* - reset GUI items of xvroutine
- *xvf_create_form()* - create and map GUI of xvroutine
- *xvf_destroy_form()* - destroy GUI of xvroutine & free associated memory
- *xvf_get_attribute()* - get a single attribute of a GUI item
- *xvf_get_attributes()* - get multiple attributes of a GUI item
- *xvf_get_xvobject()* - return desired xvobject component of kformstruct
- *xvf_remove_extra_call()* - remove function call from GUI item
- *xvf_remove_gui_callback()* - remove callback from GUI item
- *xvf_run_form()* - run the GUI of an xvroutine

• *xvf_set_attribute()* - set a single attribute of a GUI item
• *xvf_set_attributes()* - set multiple attributes of a GUI item

C. Routines for Form Creation, Display, Etc

These routines are used to create a form from a User Interface Specification (UIS) file, to display and run the form, to destroy it, and to clear its selections of old values.

C.1. xvf_create_form() — *create and map GUI of xvroutine*

Synopsis

```
kform *xvf_create_form(
   char  *filename,
   int   glyph_type,
   void (*callback)(kform *, ksubform *, kaddr),
   kaddr client_data,
   int   x,
   int   y,
   int   editable)
```

Input Arguments

`filename`
name of the UIS file

`glyph_type`
type of glyph associated with this form:

KNONE or SIMPLE

`callback`
optional callback routine for this form

`client_data`
client data for callback routine

`x`
X location at which to place the newly created GUI. If values of x and y are both negative 1, placement of the GUI must be done manually.

`y`
Y location at which to place the newly created GUI. If values of x and y are both negative 1, placement of the GUI must be done manually.

`editable`
controls use of the menuforms by the user.

XVF_FULL_EDIT: if GUI is to be completely editable, eg,
as when creating a GUI with preview or guise.
XVF_PARTIAL_EDIT: for normal use where the GUI of the
application is to be editable by the user
in ways that will not affect the performance

of the program (for example, button titles
or location).

XVF_NO_EDIT: will disable use of the menuforms altogether,
so that the user will not be able to change
any aspect of the GUI.

Returns

A pointer to the form tree on success, NULL on failure.

Description

This is the main driver for the routines that creates and maps the forms of an xvroutine, both externally (the GUI made up of objects) and internally (the abstract data structure referred to as a form tree).

C.2. xvf_run_form() — *run the GUI of an xvroutine*

Synopsis

```
void xvf_run_form(void)
```

Returns

none

Description

Runs the GUI of an xvroutine in a loop that is a modified version of XtMainLoop(). XEvents are waited on and dispatched until the number of toplevel windows is zero. The routine then returns to the calling routine.

C.3. xvf_destroy_form() — *destroy GUI of xvroutine & free associated memory*

Synopsis

```
void xvf_destroy_form(
    kform *form)
```

Input Arguments

`form`

pointer to the form tree being destroyed

Description

Destroys all GUI objects associated with a particular form tree and the form tree itself.

C.4. xvf_clear_selections() — *reset GUI items of xvroutine*

Synopsis

```
void xvf_clear_selections(
   kform *form)
```

Input Arguments

`form`

pointer to the form tree associated with the GUI to be cleared.

Description

Resets all action buttons and "live" selections on the GUI, so that they are not considered "selected" by the user. The necessary call to this routine is automatically generated by conductor in the main GUI driver of an xvroutine, located in "form_drv.c".

D. Setting & Getting GUI Item Attributes

The *xvf_set_attribute()* function allows the xvroutine to change attributes of the items on its GUI during runtime, and can be useful in a variety of situations.

For example, some changes that might be made by an application to its GUI during runtime include:

- The values displayed in the parameter boxes of InputFile, OutputFile, Integer, Float, Double, String, or StringList selections.
- The values set on Toggles, Logicals, Cycles, Lists, and Flags.
- The title of a GUI selection.
- The text displayed by Blank selections so that they may be used to display current values, parameter settings, or other pertinent information.
- Deactivation or re-activation of a particular button or selection depending on some other action by the user.

- The contents displayed in a list selection.

For these and a wide variety other reasons, *xvf_set_attribute()* can be called to change attributes of GUI items as a reaction to certain state information of the xvroutine. The following example changes the value of a list selection to the 1st element of the list:

```
xvf_set_attribute(pane_info->list_struct, XVF_LIST_INDEX, 1);
```

All candidates for the *kform_struct* first parameter are defined by the xvroutine code generator as elements of the GUI Information structure which is generated in "form_info.h" (please see Chapter 6 of the Toolbox Programmer's Manual). Pass the field in the GUI Information structure named "*var*_struct" where *var* is the variable name you provided on the UIS line that defines the GUI item that you wish to change.

Following the *kform_struct* parameter is an attribute/value pair. The supported attributes are defined by the *xvforms* library; the data type of the corresponding value is determined by the nature of the attribute.

The *xvf_get_attribute()* routine can be used to obtain the current values of the same GUI attributes. Calls to *xvf_get_attribute()*, however, are very rare. This is because there is generally no need for the xvroutine to inquire about attribute settings of GUI selections; it was, after all, the xvroutine that set the values of the attributes in the first place.

Both *xvf_set_attribute()* and *xvf_get_attribute()* have variable argument versions. For example, consider multiple sequential calls to *xvf_set_attribute()* using the same *kform_struct* pointer, as in the following:

```
xvf_set_attribute(pane_info->i_struct, XVF_TITLE, "Latest and Greatest");
xvf_set_attribute(pane_info->i_struct, XVF_FILE_NAME, "image:ball");
```

If desired, the above code could be combined into a single call to the variable argument *xvf_set_attributes()*, as in:

```
xvf_set_attributes(pane_info->i_struct,
             XVF_TITLE, "Latest and Greatest",
             XVF_FILE_NAME, "image:ball",
             NULL);
```

The two code segments are identical in effect.

Note that *xvf_set_attribute()*, *xvf_set_attribute()*, and their variable argument counterparts are *not* available to kroutines, as kroutines by definition are not linked against the *xvforms* library.

IMPORTANT NOTE

When you use *xvf_set_attribute()* to set the value of a selection, it DOES NOT MODIFY THE CORRESPONDING VARIABLE IN THE GUI INFORMATION STRUCT. You *must* set the GUI Information struct variable yourself! For example, suppose you have an integer named *n*, with a current value of 0. Then, suppose that you set the value of the integer selection to 100, using:

```
xvf_set_attribute(pane_info->n_struct, XVF_INT_VAL, 100);
```

At this point, the value displayed in the text box of the integer selection will read, "100." However, the value of "pane_info->n" will *still* be set to 0! Thus, to keep the values in the GUI Information structure in sync with the values displayed on the GUI, the next line of code should be:

```
pane_info->n = 100;
```

Neglecting to set the GUI Information structure values after calls to *xvf_set_attribute()* that change the values of a selection can cause very subtle bugs in an xvroutine.

D.1. xvf_set_attribute() — *set a single attribute of a GUI item*

Synopsis

```
xvf_set_attribute(

   kform_struct kformstruct,
   char         *attribute,
   data         value)
```

Input Arguments

`kformstruct`
: GUI item for which to set the attribute

`attribute`
: the attribute name

`value`
: the attribute value

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

sets a single attribute for a GUI item

Restrictions

Value provided must be of the expected data type

D.2. xvf_get_attribute() — *get a single attribute of a GUI item*

Synopsis

```
xvf_get_attribute(

   kform_struct kformstruct,
   char         *attribute,
   data         *value)
```

Input Arguments

`kformstruct`
GUI item for which to get the attribute

`attribute`
the attribute name

`value`
pointer to the the attribute value

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

gets a single attribute for a GUI item

Restrictions

Value pointer passed in must be of the expected data type

D.3. xvf_set_attributes() — *set multiple attributes of a GUI item*

Synopsis

```
int xvf_set_attributes(
   kform_struct *kformstruct,
   kvalist)
```

Input Arguments

`kformstruct`
the generic kform_struct associated with the GUI item (automatically generated by conductor in the GUI Information Structure) attribute - The attribute to be changed value - value to which to set attribute [attr/value]- as many additional attribute/value pairs as desired NULL - attribute/value pairs must be ended with NULL

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

Sets attributes associated with GUI items and UIS lines of GUI items supported by the xvforms library. Note that the variable argument list of attribute/pairs MUST be terminated by NULL.

D.4. xvf_get_attributes() — *get multiple attributes of a GUI item*

Synopsis

```
int xvf_get_attributes(
   kform_struct *kformstruct,
   kvalist)
```

Input Arguments

`kformstruct`

the generic kform_struct associated with the GUI item (automatically generated by conductor in the GUI Information Structure) attribute - The attribute to be changed value - value to which to set attribute [attr/value]- as many additional attribute/value pairs as desired NULL - attribute/value pairs must be ended with NULL

Returns

TRUE (1) on success, FALSE (0) otherwise

Description

Gets attributes associated with GUI items and UIS lines of GUI objects supported by the xvforms library.

E. GUI Item Resource Set

E.1. Complete GUI Item Resource Listing

Complete GUI Item Resource Listing	
Attribute	**Described In Section**
XVF_ACTIVATE	General GUI Item Attributes
XVF_BUTTONHEIGHT	General GUI Item Attributes
XVF_BUTTONTITLE	General GUI Item Attributes
XVF_BUTTONWIDTH	General GUI Item Attributes
XVF_BUTTONX	General GUI Item Attributes
XVF_BUTTONY	General GUI Item Attributes
XVF_CLIENTDATA	General GUI Item Attributes
XVF_CYCLE_ADD	Attributes of Cycles

Complete GUI Item Resource Listing	
Attribute	**Described In Section**
XVF_CYCLE_CONTENTS	Attributes of Cycles
XVF_CYCLE_DELETE	Attributes of Cycles
XVF_CYCLE_DELETEALL	Attributes of Cycles
XVF_CYCLE_INDEX	Attributes of Cycles
XVF_CYCLE_LABEL	Attributes of Cycles
XVF_CYCLE_SIZE	Attributes of Cycles
XVF_CYCLE_START	Attributes of Cycles
XVF_CYCLE_VAL	Attributes of Cycles
XVF_DELETE	General GUI Item Attributes
XVF_DESCRIPTION	General GUI Item Attributes
XVF_DISPLAY_PANE	Attributes for Subform And Pane Display
XVF_DISPLAY_SUBFORM	Attributes for Subform And Pane Display
XVF_DOUBLE_DEF	Attributes of Doubles
XVF_DOUBLE_LOWER	Attributes of Doubles
XVF_DOUBLE_PREC	Attributes of Doubles
XVF_DOUBLE_UPPER	Attributes of Doubles
XVF_DOUBLE_VAL	Attributes of Doubles
XVF_FILE_CHECK	Attributes of Input & Output Files
XVF_FILE_DEF	Attributes of Input & Output Files
XVF_FILE_NAME	Attributes of Input & Output Files
XVF_FLOAT_DEF	Attributes of Floats
XVF_FLOAT_LOWER	Attributes of Floats
XVF_FLOAT_PREC	Attributes of Floats
XVF_FLOAT_UPPER	Attributes of Floats
XVF_FLOAT_VAL	Attributes of Floats
XVF_GUIDEPANETITLE	Attributes of Floats
XVF_HEIGHT	General GUI Item Attributes
XVF_HELPPATH	Attributes of Help Buttons
XVF_INT_DEF	Attributes of Integers
XVF_INT_LOWER	Attributes of Integers
XVF_INT_UPPER	Attributes of Integers
XVF_INT_VAL	Attributes of Integers
XVF_LIST_ADD	Attributes of Lists
XVF_LIST_CONTENTS	Attributes of Lists
XVF_LIST_DELETE	Attributes of Lists
XVF_LIST_DELETEALL	Attributes of Lists
XVF_LIST_INDEX	Attributes of Lists
XVF_LIST_LABEL	Attributes of Lists
XVF_LIST_SIZE	Attributes of Lists
XVF_LIST_START	Attributes of Lists
XVF_LIST_VAL	Attributes of Lists
XVF_LITERAL	General GUI Item Attributes
XVF_LIVE	General GUI Item Attributes
XVF_LOGIC_0LABEL	Attributes of Logicals
XVF_LOGIC_1LABEL	Attributes of Logicals
XVF_LOGIC_DEF	Attributes of Logicals

Complete GUI Item Resource Listing	
Attribute	**Described In Section**
XVF_LOGIC_VAL	Attributes of Logicals
XVF_MECHANISM	Attributes of Integers, Floats, & Doubles
XVF_OPTIONAL	General GUI Item Attributes
XVF_OPTSEL	General GUI Item Attributes
XVF_PRINT_PANE	Attributes For Printing UIS Files
XVF_PRINT_SUBFORM	Attributes For Printing UIS Files
XVF_PRINT_UIS	Attributes For Printing UIS Files
XVF_ROUTINE	Attributes of Routine Buttons
XVF_STRING_DEF	Attributes of Strings
XVF_STRING_MULTILINE	Attributes of Strings
XVF_STRING_VAL	Attributes of Strings
XVF_TITLE	General GUI Item Attributes
XVF_TOGGLE_NUM	Attributes of Toggles
XVF_TOGGLE_SIZE	Attributes of Toggles
XVF_TOGGLE_TYPE	Attributes of Toggles
XVF_TOGGLE_VAL	Attributes of Toggles
XVF_VARIABLE	General GUI Item Attributes
XVF_WIDTH	General GUI Item Attributes
XVF_X	General GUI Item Attributes
XVF_XPOS	General GUI Item Attributes
XVF_Y	General GUI Item Attributes
XVF_YPOS	General GUI Item Attributes

E.2. General GUI Item Attributes

The following is a listing of general GUI item attributes. These are attributes which are common to a large number of GUI items.

Descriptions of General Xvforms Attributes	
Attribute	**Description**
XVF_ACTIVATE	This attribute, which may be set to TRUE (1) or FALSE (0), indicates whether an item on the GUI is active or inactive. If an item is inactive, it will still be displayed on the GUI; however, it will not accept any input. It may appear undersized, fuzzy, and text may be difficult to read. GUI selections may be de-activated by setting Activate to FALSE (0). De-activating a guide button will eliminate access to its associated pane, and de-activating a subform button will eliminate access to its associated subform. This attribute has no effect on CLUI arguments.

Descriptions of General Xvforms Attributes	
Attribute	**Description**
XVF_BUTTONHEIGHT	This attribute specifies the height of a button in characters. It is only applicable to GUI items consisting of a single button (such as help buttons, routine buttons, subform buttons, guide buttons, and so on). For any GUI button besides subform buttons and guide buttons, this attribute is interchangeable with the Height. However, on a subform, the ButtonHeight refers to the height of the subform button while the Height specifies the height of the subform itself. Similarly for panes, the ButtonHeight refers to the height of the guide button, while the Height specifies the height of the pane itself.
XVF_BUTTONTITLE	This attribute specifies the title on a button. It is only applicable to GUI items consisting of a single button (such as help buttons, routine buttons, subform buttons, guide buttons, and so on). For any GUI button besides subform buttons and guide buttons, this attribute is interchangeable with the Title. However, on a subform, the ButtonTitle refers to the title on the subform button while the Title specifies the title displayed on the subform itself. Similarly for panes, the ButtonTitle refers to the title on the guide button, while the Title specifies the title displayed on the pane itself.
XVF_BUTTONWIDTH	This attribute specifies the width of a button in characters. It is only applicable to GUI items consisting of a single button (such as help buttons, routine buttons, subform buttons, guide buttons, and so on). For any GUI button besides subform buttons and guide buttons, this attribute is interchangeable with the Width. However, on a subform, the ButtonWidth refers to the width of the subform button while the Width specifies the width of the subform itself. Similarly for panes, the ButtonWidth refers to the width of the guide button, while the Width specifies the width of the pane itself.
XVF_BUTTONX	This attribute is used to position a button horizontally on its backplane. The X location of the button is specified floating point character widths. It is only applicable to GUI items consisting of a single button (such as help buttons, routine buttons, subform buttons, and guide buttons). For any GUI button besides subform buttons and guide buttons, this attribute is interchangeable with X. However, on a subform, the ButtonX refers to the X location of the subform button, as opposed to the location of the subform itself. For panes, ButtonX refers to the location of the guide button, while X specifies the location of the pane on its backplane.

Descriptions of General Xvforms Attributes	
Attribute	**Description**
XVF_BUTTONY	This attribute is used to position a button vertically on its backplane. The Y location of the button is specified floating point character widths. It is only applicable to GUI items consisting of a single button (such as help buttons, routine buttons, subform buttons, and guide buttons). For any GUI button besides subform buttons and guide buttons, this attribute is interchangeable with Y. However, on a subform, the ButtonY refers to the Y location of the subform button, as opposed to the location of the subform itself. For panes, ButtonY refers to the location of the guide button, while Y specifies the location of the pane on its backplane.
XVF_CLIENTDATA	This attribute allows you to set and get a client_data pointer that is associated with the GUI item specified. By default, the client_data is NULL; however, you may set it to any pointer as desired, and retrieve it for use when desired. See Program Services Volume I, Chapter 2, Section L.6 for an explanation on the use of client_data pointers in general.
XVF_DELETE	This is an *action* attribute; that is, it can only be used with *xvf_set_attribute(s)()*, and should be passed a value of TRUE. When used, it will delete the GUI item specified.
XVF_DESCRIPTION	This attribute accommodates a brief description of the purpose of the selection. Used only by the CLUI, comments in auto-generated code are taken directly from the description field. Descriptions should be clear and concise.
XVF_GUIDEPANE_TITLE	This attribute specifies the title of the Guidepane on subforms with guidepanes (ie, those subforms with multiple panes and a set of guide buttons). It is only specified for subforms, and is distinct from the Title of the subform, which can be specified separately.
XVF_GUIDEPANE_TITLE_XPOS	This attribute is used to position the Title of the guide pane within the subform. The X offset is specified in floating point character widths, from the upper left hand corner of the GUI item.
XVF_GUIDEPANE_TITLE_YPOS	This attribute is used to position the Title of the guide pane within the subform. The Y offset is specified in floating point character heights, from the upper left hand corner of the GUI item.
XVF_HEIGHT	The Height attribute is used to size the GUI item. The overall height of the entire GUI item is specified in floating point character heights. To tack the GUI item to the right side of its backplane, a value of -1.0 is used. InputFiles, OutputFiles, Integers, Floats, Doubles, Flags, and Blanks are internally restricted to a Height of 1.0; thus, setting the Height attribute on one of these selections will have no effect. Strings and StringLists can have a height more than 1.0; however, the XVF_STRING_MULTILINE attribute must be set to TRUE (1) prior to setting the height.

Descriptions of General Xvforms Attributes	
Attribute	**Description**
XVF_HEIGHT_TACKED_SEL	If desired, the HeightTacked attribute may be used instead of the Height attribute to size the GUI item vertically. When the height of the GUI item is tacked, the item will appear at the location specified by the X and Y attributes, but the bottom will be pinned, or "tacked" to the bottom of the backplane of the GUI item, so that the height of the GUI item depends on its location and the height of its backplane. This is especially useful with InputFile, OutputFile, Workspace, and other GUI items as it allows them to grow and shrink when the backplane is resized by the user with the window manager. Note that setting WidthTacked to TRUE is identical to setting Width to -1.0. Note that setting HeightTacked to TRUE is identical to setting Height to -1.0. InputFiles, OutputFiles, Integers, Floats, Doubles, Flags, and Blanks are internally restricted to a Height of 1.0; thus, setting this attribute on one of these selections will have no effect. Strings and StringLists can have a height more than 1.0; however, the XVF_STRING_MULTILINE attribute must be set to TRUE (1) prior to setting height tacking.
XVF_LITERAL	This attribute stores the *literal* string value of the current value for an InputFile, OutputFile, Integer, Float, Double, String, or StringList selection (ie, all selections with a text parameter box). For example, if the value 8.76 appears in the text parameter box of a Double selection, the value of the double is 8.76; however, the Literal value of the double is the string, "8.76". It is necessary to have literals so that the expression parser can be used with various types of selections. For example, in VisiQuest, the user might enter the string "(i+10)/j" as the value for an Integer selection. The value of the integer will be calculated according to the values of i and j; if i was currently set to 2 and j was currently set to 6, the integer value of the expression would be 2. However, the literal value of the integer selection would be "(i+10)/j"; this string needs to be saved so that VisiQuest can properly save and restore workspaces. In general, the Literal attribute should not be set by the application; however, it may be retrieved for inspection if desired.

Descriptions of General Xvforms Attributes	
Attribute	**Description**
XVF_LIVE	This attribute, which may be set to TRUE (1) or FALSE (0), specifies whether or not a selection is "live". When a selection is "live", software control is immediately returned to the application program when the user initiates a change of value in the selection. A change of value can be initiated in the following ways: 1) When the user hits <CR> in the text parameter box of a "live" InputFile, OutputFile, Integer, Float, Double, String, or StringList selection 2) When the user moves a scroll bar associated with a "live" Integer, Float, or Double selection 3) When the user chooses a new value for a Toggle, Logical, Flag, List, DisplayList, StringList or Cycle selection In the case of a kroutine, making a selection "live" means that when a change of value is initiated on the GUI in VisiQuest, the network involving the glyph associated with that kroutine will be re-run from that point on. In the case of an xvroutine, making a selection "live" means that when a change of value is initiated on the xvroutine's GUI, software control will be passed immediately back to the xvroutine. Thus, the xvroutine may take appropriate action according to the value of the selection. An example is found in the GUI of the guise design tool. Here, the Input-File selection for the "Input UIS File" option is "live". Thus, when the user enters the name of a new input UIS file and hits <cr>, software control is returned to the guise application, where the routine that handles the input of a new UIS file is immediately called. Note that there is only *one* piece of information necessary to complete the input action - a new UIS filename. This makes the "Input UIS File" selection an ideal choice to be "live". When a selection is not "live" (FALSE), software flow will not be returned to the application until the user clicks on an action button, or until the user changes the value of another selection, which happens to be "live". If more than one piece of information is needed before action can be initiated, however, individual selections should NOT be "live".

Descriptions of General Xvforms Attributes	
Attribute	**Description**
XVF_OPTIONAL	This attribute, which may be set to TRUE (1) or FALSE (0), indicates whether a GUI selection or CLUI argument is optional or required. On the GUI, a selection that is optional will have a small box to the left of the title of the selection. The user may click on the box (highlighting it) to indicate that they wish to use the value specified for the selection. On the CLUI, an argument that is optional may be omitted from the command line, in which case it will take on the default value.
XVF_OPTSEL	The "Optional Selected" attribute affects both the GUI and the CLUI, and may take on one of three values: 0, 1, or 2. When a selection is not optional, this field *must* be set to 1. When a selection is optional, it may have any of the three values, which have meanings as follows: a) 0 -- NOT SELECTED On both the GUI and the CLUI, a value of NOT SELECTED (0) indicates that the default action is *not to use* the value of the optional selection at all, or to use the default value of the selection. On the GUI, this is visually represented by the optional box of the selection being unhighlighted. b) 1 -- SELECTED On both the GUI and the CLUI, a value of SELECTED (1) indicates that the default action is *to use* the value of the optional selection, whether it be the default value or a new value provided by the user. On the GUI, this is visually represented by the optional box of the selection being highlighted. c) 2 -- SELECTED, BUT NOT SHOWN A value of SELECTED, BUT NOT SHOWN (2) causes the selection to act as if the attribute was set to SELECTED (1). However, the optional box for the selection will not appear on the GUI. When a kroutine has many optional CLUI arguments, the optional boxes that appear on the pane for VisiQuest GUI may be considered bulky or unsightly; this setting provides a mechanism for simplifying the GUI. Note: You are NOT allowed to set a member of a group to this value.
XVF_TITLE	The Title attribute specifies the string that will be displayed on the GUI to identify a selection. The title may contain spaces, but should remain fairly short to be effective. Titles may be left blank. If a very long title is needed, it is recommended that you use a blank selection to hold the text separately, and leave the selection itself with no title.

Descriptions of General Xvforms Attributes	
Attribute	**Description**
XVF_VARIABLE	This attribute provides a unique name for the variable associated with a GUI item or CLUI argument. In code generated for the command line user interface of any VisiQuest program, the variable field becomes the name of the argument. In code generated for the graphical user interface of an xvroutine, the variable field is translated into the naming conventions for the GUI Information structure that provides the link between the application program and its GUI.
XVF_WIDTH	The Width attribute is used to size the GUI item; it specifies the overall width of the entire GUI item in floating point character widths. To tack the GUI item to the right side of its backplane, a value of -1.0 is used.
XVF_WIDTH_TACKED_SEL	If desired, the WidthTacked attribute may be used instead of the Width attribute to size the GUI item vertically. When the width of the GUI item is tacked, the item will appear at the location specified by the X and Y attributes, but the right side will be pinned, or "tacked" to the right side of the backplane of the GUI item, so that the width of the GUI item depends on its location and the width of its backplane. This is especially useful with InputFile, OutputFile, Workspace, and other GUI items as it allows them to grow and shrink when the backplane is resized by the user with the window manager. Note that setting WidthTacked to TRUE is identical to setting Width to -1.0.
XVF_X	This attribute is used to position the GUI item horizontally on its backplane. The X location of the GUI item is specified floating point character widths.
XVF_XPOS	This attribute is used to position the Title of the GUI item within the GUI item itself. The X offset is specified in floating point character widths, from the upper left hand corner of the GUI item.
XVF_Y	This attribute is used to position the GUI item vertically on its backplane. The Y location of the GUI item is specified floating point character heights.
XVF_YPOS	This attribute is used to position the Title of the GUI item within the GUI item itself. The Y offset is specified in floating point character heights, from the upper left hand corner of the GUI item.

Summary of General Xvforms Attributes			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_ACTIVATE	int	Any GUI items *except*: Blanks Workspaces	TRUE/FALSE

Summary of General Xvforms Attributes			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_BUTTONHEIGHT	float	Any GUI button, including: ActionButtons GuideButtons (panes) HelpButtons QuitButtons RoutineButtons SubformButtons (subforms)	height > 0.0 (in characters)
XVF_BUTTONTITLE	char *	Any GUI button, including: ActionButtons GuideButtons (panes) HelpButtons QuitButtons RoutineButtons SubformButtons (subforms)	any string
XVF_BUTTONWIDTH	float	Any GUI button, including: ActionButtons GuideButtons (panes) HelpButtons QuitButtons RoutineButtons SubformButtons (subforms)	width > 0.0 (in characters)
XVF_BUTTONX	float	Any GUI button, including: ActionButtons GuideButtons (panes) HelpButtons QuitButtons RoutineButtons SubformButtons (subforms)	x > 0.0 (in characters)

Summary of General Xvforms Attributes			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_BUTTONY	float	Any GUI button, including: ActionButtons GuideButtons (panes) HelpButtons QuitButtons RoutineButtons SubformButtons (subforms)	y > 0.0 (in characters)
XVF_CLIENTDATA	kaddr	Any GUI item *except* Form	pointer to desired client_data
XVF_DELETE	int	Any GUI items *except*: Forms GuidePanes	TRUE (Action Attribute)
XVF_DESCRIPTION	char *	ActionButtons Cycles DisplayLists Doubles Flags Floats HelpButtons InputFiles Integers Lists Logicals OutputFiles RoutineButtons Strings StringLists Toggles Workspaces	any string
XVF_GUIDEPANE_TITLE	char *	Subforms having a Guidepane	any string
XVF_GUIDEPANE_TITLE_XPOS	float	Subforms having a Guidepane	x >= 0.0
XVF_GUIDEPANE_TITLE_YPOS	float	Subforms having a Guidepane	y >= 0.0
XVF_HEIGHT	int	Any GUI items *except*: Blanks	height > 0.0 (in characters)

Summary of General Xvforms Attributes			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_HEIGHT_TACKED_SEL	int	Any GUI items *except*: Blanks	TRUE/FALSE
XVF_LITERAL	int	Doubles Floats InputFiles Integers OutputFiles Strings StringLists Toggles	any string
XVF_LIVE	int	Cycles DisplayLists Doubles Flags Floats InputFiles Integers Lists Logicals OutputFiles Strings StringLists Toggles	TRUE/FALSE
XVF_OPTIONAL	int	Cycles DisplayLists Doubles Flags Floats InputFiles Integers Lists Logicals OutputFiles Strings StringLists Toggles	TRUE/FALSE

Summary of General Xvforms Attributes			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_OPTSEL	int	Cycles DisplayLists Doubles Flags Floats InputFiles Integers Lists Logicals OutputFiles Strings StringLists Toggles	TRUE/FALSE
XVF_TITLE	char *	Any GUI item	any string
XVF_VARIABLE	int	Any GUI item	any string that will not cause C compiler syntax errors
XVF_WIDTH	float	Any GUI items *except*: Blanks	width > 0.0 (in characters)
XVF_WIDTH_TACKED_SEL	int	Any GUI items *except*: Blanks	TRUE/FALSE
XVF_X	float	Any GUI items *except*: Blanks	x >= 0.0 (in characters)
XVF_XPOS	float	Forms GuidePanes Panes Subforms Toggles Blanks	xpos >= 0.0
XVF_Y	float	Any GUI items *except*: Blanks	y >= 0.0 (in characters)
XVF_YPOS	float	Any GUI items *except*: Blanks	ypos >= 0.0 (in characters)

E.3. Attributes of InputFiles and OutputFiles

The following is a listing of GUI item attributes that only apply to InputFile and OutputFile selections.

Descriptions of Attributes For InputFile & OutputFile Selections	
Attribute	**Description**
XVF_FILE_CHECK	When set to TRUE (1), this attribute causes filenames entered by the user into InputFile selections to be checked for existence and read permission; if a file entered does not exist or is not readable, an error message is issued. For OutputFile selections, the file entered by the user will be checked for write permission; if the file entered is not writable, an error message is issued. To prevent InputFile and OutputFile selections from checking validity of files, set this attribute to FALSE (0).
XVF_FILE_DEF	The FileDefault attribute specifies a default filename to be used with an InputFile or OutputFile selection. The default filename may be an alias, as specified in the $TOOLBOX/repos/Aliases file, such as "image:ball". Alternatively, it may be the full path to a file. Tildas are expanded appropriately. The $TOOLBOX variable may be used as part of the path. If no full path, but only a file name, is specified, the file is considered to exist in the local "." directory. However, omission of the path is generally not a good idea, as one cannot predict where in the directory structure a user will execute the application.
XVF_FILE_NAME	The FileName attribute specifies the current value of the filename to be used with an InputFile, OutputFile, AnswerInputFile, or AnswerOutputFile selection. The filename may be an alias, as specified in the $TOOLBOX/repos/Aliases file, or may be the full path to a file. The $TOOLBOX variable may be used as part of the path. If no full path is specified, the file is considered to exist in the local "." directory.

Summary of Attributes For InputFile & OutputFile Selections			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_FILE_CHECK	int	InputFiles, OutputFiles	TRUE/FALSE
XVF_FILE_DEF	char *	InputFiles, OutputFiles	any file name
XVF_FILE_NAME	char *	InputFiles, OutputFiles	any file name

E.4. Attributes of Logicals

The following is a listing of GUI item attributes that apply to Logical selections.

Descriptions of Attributes For Logical Selections	
Attribute	**Description**
XVF_LOGIC_0LABEL	This is the label that will appear on the value button of the logical selection when the value of the logical is set to 0 (FALSE). The most commonly used label 0 strings are "False" and "No".
XVF_LOGIC_1LABEL	This is the label that will appear on the value button of the logical selection when the value of the logical is set to 1 (TRUE). The most commonly used label 1 strings are "True" and "Yes".
XVF_LOGIC_DEF	This attribute specifies the default value to be used with a Logical selection. It may be set to TRUE (1) or FALSE (0).
XVF_LOGIC_VAL	This attribute specifies the current value of a Logical selection. It may be set to TRUE (1) or FALSE (0).

Summary of Attributes For Logical Selections			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_LOGIC_0LABEL	char *	Logicals	Any string
XVF_LOGIC_1LABEL	char *	Logicals	Any string
XVF_LOGIC_DEF	int	Logicals	0 or 1
XVF_LOGIC_VAL	int	Logicals	0 or 1

E.5. Attributes of Integers

The following is a listing of GUI item attributes that only apply to Integer selections.

Descriptions of Attributes For Integer Selections	
Attribute	**Description**
XVF_INT_DEF	This attribute specifies the default value to be used with an Integer selection. The values of the Lower and Upper bounds determine legal values for the integer default, which may be unbounded, bounded between a lower and an upper value, strictly less than zero, less than or equal to zero, strictly greater than zero, or greater than or equal to zero.

Descriptions of Attributes For Integer Selections	
Attribute	**Description**
XVF_INT_LOWER	This attribute is used to specify the lower bound of the legal values for a bounded Integer selection. It is used in conjunction with the Upper bound to specify a integer that is unbounded, strictly less than zero, less than or equal to zero, strictly greater than zero, or greater than or equal to zero. The following rules apply: Lower = Upper = -2 { legal values < 0 } Lower = Upper = -1 { legal values <= 0 } Lower = Upper = 1 { legal values >= 0 } Lower = Upper = 2 { legal values > 0 } Lower = Upper = X { no range checking } Lower = X; Upper = Y { X <= legal value <= Y }
XVF_INT_UPPER	This attribute is used to specify the upper bound of the legal values for a bounded Integer selection. It is used in conjunction with the Lower bound to specify a integer that is unbounded, strictly less than zero, less than or equal to zero, strictly greater than zero, or greater than or equal to zero. The following rules apply: Lower = Upper = -2 { legal values < 0 } Lower = Upper = -1 { legal values <= 0 } Lower = Upper = 1 { legal values >= 0 } Lower = Upper = 2 { legal values > 0 } Lower = Upper = X { no range checking } Lower = X; Upper = Y { X <= legal value <= Y }
XVF_INT_VAL	The IntegerValue attribute specifies the current value of a Integer selection. The values of the lower and upper bounds determine legal values for the integer value, which may be unbounded, bounded between a lower and an upper value, strictly less than zero, less than or equal to zero, strictly greater than zero, or greater than or equal to zero.
XVF_MECHANISM	The Mechanism attribute is used to specify whether or not a scrollbar will be used with integers, floats, or doubles that are strictly bound (ie, the lower bound is not equal to the upper bound). When integers, floats, or doubles are strictly bound and the width is sufficient, a scroll bar will appear to the right of the parameter box, allowing the user to set the value with the scroll bar. To turn off the scrollbar, provide a value of 0; to enable the scrollbar, provide a value of 1. In the future, specifying a 2 may indicate the preference for an alternate mechanism to be used for this purpose, such as a dial.

Summary of Attributes For Integer Selections			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_INT_DEF	int	Integers	Any integer value, within bounds specified by XVF_INT_LOWER and XVF_INT_UPPER

Summary of Attributes For Integer Selections			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_INT_LOWER	int	Integers	lower < XVF_INT_UPPER
XVF_INT_UPPER	int	Integers	upper > XVF_INT_LOWER
XVF_INT_VAL	int	Integers	Any integer value, within bounds specified by XVF_INT_LOWER and XVF_INT_UPPER
XVF_MECHANISM	int	Integers Floats Doubles	0 or 1

E.6. Attributes of Floats

The following is a listing of GUI item attributes that only apply to Float selections.

Descriptions of Attributes For Float Selections	
Attribute	**Description**
XVF_FLOAT_DEF	The FloatDefault attribute specifies a default value to be used with a Float selection. The values of the lower and upper bounds determine legal values for the float default, which may be unbounded, bounded between a lower and an upper value, strictly less than zero, less than or equal to zero, strictly greater than zero, or greater than or equal to zero.
XVF_FLOAT_LOWER	This attribute is used to specify the lower bound of the legal values for a bounded Float selection. It is used in conjunction with the upper bound to specify a float that is unbounded, strictly less than zero, less than or equal to zero, strictly greater than zero, or greater than or equal to zero. The following rules apply: Lower = Upper = -2.0 { legal values < 0.0 } Lower = Upper = -1.0 { legal values <= 0.0 } Lower = Upper = 1.0 { legal values >= 0.0 } Lower = Upper = 2.0 { legal values > 0.0 } Lower = Upper = X { no range checking } Lower = X; Upper = Y { X <= legal value <= Y }
XVF_FLOAT_PREC	The FloatPrecision attribute specifies the number of decimal places that are presented to the user with the Float selection. If the FloatPrecision value is zero, the float value will be presented to the user with the format "%g"; ie, trailing zeroes are removed from the result, and a decimal point will appear only if it is followed by a digit. If the FloatPrecision value is greater than zero, the float value will be presented with that number of significant digits. The maximum number of significant digits for a float selection is 7.

Descriptions of Attributes For Float Selections	
Attribute	**Description**
XVF_FLOAT_UPPER	This attribute is used to specify the upper bound of the legal values for a bounded Float selection. It is used in conjunction with the Lower bound to specify a float that is unbounded, strictly less than zero, less than or equal to zero, strictly greater than zero, or greater than or equal to zero. The following rules apply: Lower = Upper = -2.0 { legal values < 0.0 } Lower = Upper = -1.0 { legal values <= 0.0 } Lower = Upper = 1.0 { legal values >= 0.0 } Lower = Upper = 2.0 { legal values > 0.0 } Lower = Upper = X { no range checking } Lower = X; Upper = Y { X <= legal value <= Y }
XVF_FLOAT_VAL	The FloatValue attribute specifies the current value of a Float selection. The values of the lower and upper bounds determine legal values for the float value, which may be unbounded, bounded between a lower and an upper value, strictly less than zero, less than or equal to zero, strictly greater than zero, or greater than or equal to zero.
XVF_MECHANISM	The Mechanism attribute is used to specify whether or not a scrollbar will be used with integers, floats, or doubles that are strictly bound (ie, the lower bound is not equal to the upper bound). When integers, floats, or doubles are strictly bound and the width is sufficient, a scroll bar will appear to the right of the parameter box, allowing the user to set the value with the scroll bar. To turn off the scrollbar, provide a value of 0; to enable the scrollbar, provide a value of 1. In the future, specifying a 2 may indicate the preference for an alternate mechanism to be used for this purpose, such as a dial.

Summary of Attributes For Float Selections			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_FLOAT_DEF	float	Floats	Any float value, within bounds specified by XVF_FLOAT_LOWER and XVF_FLOAT_UPPER
XVF_FLOAT_LOWER	float	Floats	lower < XVF_FLOAT_UPPER
XVF_FLOAT_PREC	float	Floats	0 <= precision <= 7
XVF_FLOAT_UPPER	float	Floats	upper > XVF_FLOAT_LOWER
XVF_FLOAT_VAL	float	Floats	Any float value, within bounds specified by XVF_FLOAT_LOWER and XVF_FLOAT_UPPER
XVF_MECHANISM	int	Integers Floats Doubles	0 or 1

E.7. Attributes of Doubles

The following is a listing of GUI item attributes that only apply to Double selections.

Descriptions of Attributes For Double Selections

Attribute	Description
XVF_DOUBLE_DEF	The DoubleDefault attribute specifies a default value to be used with a Double selection. The values of the lower and upper bounds determine legal values for the double default, which may be unbounded, bounded between a lower and an upper value, strictly less than zero, less than or equal to zero, strictly greater than zero, or greater than or equal to zero.
XVF_DOUBLE_LOWER	This attribute is used to specify the lower bound of the legal values for a bounded Double selection. It is used in conjunction with the upper bound to specify a double that is unbounded, strictly less than zero, less than or equal to zero, strictly greater than zero, or greater than or equal to zero. The following rules apply: Lower = Upper = -2.0 { legal values < 0.0 } Lower = Upper = -1.0 { legal values <= 0.0 } Lower = Upper = 1.0 { legal values >= 0.0 } Lower = Upper = 2.0 { legal values > 0.0 } Lower = Upper = X { no range checking } Lower = X; Upper = Y { X <= legal value <= Y }
XVF_DOUBLE_PREC	The DoublePrecision attribute specifies the number of decimal places that are presented to the user with the Double selection. If the DoublePrecision value is zero, the double value will be presented to the user with the format "%g"; ie, trailing zeroes are removed from the result, and a decimal point will appear only if it is followed by a digit. If the DoublePrecision value is greater than zero, the double value will be presented with that number of significant digits. The maximum number of significant digits for a double selection is 14.
XVF_DOUBLE_UPPER	This attribute is used to specify the upper bound of the legal values for a bounded Double selection. It is used in conjunction with the lower bound to specify a double that is unbounded, strictly less than zero, less than or equal to zero, strictly greater than zero, or greater than or equal to zero. The following rules apply: Lower = Upper = -2.0 { legal values < 0.0 } Lower = Upper = -1.0 { legal values <= 0.0 } Lower = Upper = 1.0 { legal values >= 0.0 } Lower = Upper = 2.0 { legal values > 0.0 } Lower = Upper = X { no range checking } Lower = X; Upper = Y { X <= legal value <= Y }
XVF_DOUBLE_VAL	The DoubleValue attribute specifies the current value of a Double selection. The values of the lower and upper bounds determine legal values for the double value, which may be unbounded, bounded between a lower and an upper value, strictly less than zero, less than or equal to zero, strictly greater than zero, or greater than or equal to zero.

Descriptions of Attributes For Double Selections	
Attribute	**Description**
XVF_MECHANISM	The Mechanism attribute is used to specify whether or not a scrollbar will be used with integers, floats, or doubles that are strictly bound (ie, the lower bound is not equal to the upper bound). When integers, floats, or doubles are strictly bound and the width is sufficient, a scroll bar will appear to the right of the parameter box, allowing the user to set the value with the scroll bar. To turn off the scrollbar, provide a value of 0; to enable the scrollbar, provide a value of 1. In the future, specifying a 2 may indicate the preference for an alternate mechanism to be used for this purpose, such as a dial.

Summary of Attributes For Double Selections			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_DOUBLE_DEF	double	Doubles	Any double value, within bounds specified by XVF_DOUBLE_LOWER and XVF_DOUBLE_UPPER
XVF_DOUBLE_LOWER	double	Doubles	lower < XVF_DOUBLE_UPPER
XVF_DOUBLE_PREC	double	Doubles	0 <= precision <= 10
XVF_DOUBLE_UPPER	double	Doubles	upper > XVF_DOUBLE_LOWER
XVF_DOUBLE_VAL	double	Doubles	Any double value, within bounds specified by XVF_DOUBLE_LOWER and XVF_DOUBLE_UPPER
XVF_MECHANISM	int	Integers Floats Doubles	0 or 1

E.8. Attributes of Strings

The following is a listing of GUI item attributes that apply to String and StringList selections.

Descriptions of Attributes For String & StringList Selections	
Attribute	**Description**
XVF_STRING_DEF	The StringDefault attribute specifies a default value to be used with a String or StringList selection. The string may consist of any printable ascii characters.
XVF_STRING_MULTILINE	The StringMultiLine attribute specifies whether or not a string or stringlist selection should have a text parameter box that is taller than 1 character height. If TRUE (1), the string or stringlist selection will be allowed to be taller than a single character height.

Descriptions of Attributes For String & StringList Selections	
Attribute	**Description**
XVF_STRING_VAL	The StringValue attribute specifies the current value of a String or StringList selection. The string may consist of any printable ascii characters.

Summary of Attributes For String & StringList Selections			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_STRING_DEF	char *	Strings StringLists	any string
XVF_STRING_MULTILINE	int	Strings StringLists	TRUE/FALSE
XVF_STRING_VAL	char *	Strings StringLists	any string

E.9. Attributes of Toggles

The following is a listing of GUI item attributes that apply to Toggle selections.

Descriptions of Attributes For Toggle Selections	
Attribute	**Description**
XVF_TOGGLE_NUM	The ToggleNum attribute specifies the index (starting at 1) of the currently selected toggle member. For example, if the third member in the toggle is selected, the toggle num will have a value of 3.
XVF_TOGGLE_SIZE	The ToggleSize attribute specifies the number of members in the toggle. At this point, it restricted to being a READ-ONLY attribute; ie, it can only be used with *xvf_get_attribute(s)()*, and cannot be used to change the size of the toggle during execution.
XVF_TOGGLE_TYPE	This attribute specifies the data type of the toggle. Supported toggle types include InputFiles, OutputFiles, Integers, Logicals, Flags, Floats, and Strings. This is a READ-ONLY attribute; ie, it can only be used with *xvf_get_attribute(s)()*, and cannot be used to change the data type of a toggle during runtime.

Descriptions of Attributes For Toggle Selections

Attribute	Description

XVF_TOGGLE_VAL	The ToggleVal attribute is the current value of the toggle. Since the data type of the toggle may vary, the value is always stored in a string buffer; to convert the string value to the proper data type, *atoi()* or *atof()* may be used in the case of Flag, Logical, Integer, Float, and Double toggles. For toggles of flags and logicals, the value returned will be the number (in order of appearance) of the flag or logical toggle member that is currently selected. Therefore, in a toggle of flags or logicals, if the Nth member is selected, the toggle will take on a value of N. Note that in the case of flag and logical toggles, the ToggleVal and ToggleNum attributes are the same. For toggles of integers, floats, doubles, strings, input files, and output files, the value returned will be the default value specified for the currently selected toggle member. The following examples illustrate the setting and getting of toggle values.
	
	`int ival;`
	`float fval;`
	`double dval;`
	`char buffer[KLENGTH];`
	
	`/* set value of flag, logical, or integer toggle */`
	`ksprintf(buffer, "%d", ival);`
	`xvf_set_attribute(kformstruct, XVF_TOGGLE_VAL,`
	`buffer);`
	
	`/* get value of flag, logical, or integer toggle */`
	`xvf_get_attribute(kformstruct, XVF_TOGGLE_VAL,`
	`&buffer);`
	`ival = atoi(buffer);`
	
	`/* set value of float toggle */`
	`ksprintf(buffer, "%f", fval);`
	`xvf_set_attribute(kformstruct, XVF_TOGGLE_VAL,`
	`buffer);`
	
	`/* get value of float toggle */`
	`xvf_get_attribute(kformstruct, XVF_TOGGLE_VAL,`
	`&buffer);`
	`fval = (float) atof(buffer);`
	
	`/* set value of double toggle */`
	`ksprintf(buffer, "%g", dval);`
	`xvf_set_attribute(kformstruct, XVF_TOGGLE_VAL,`
	`buffer);`
	
	`/* get value of double toggle */`
	`xvf_get_attribute(kformstruct, XVF_TOGGLE_VAL,`
	`&buffer);`
	`dval = atof(buffer);`

Summary of Attributes For Toggle Selections			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_TOGGLE_NUM	int	Toggles	1 <= toggle_num <= XVF_TOGGLE_MEMBER_NUM
XVF_TOGGLE_SIZE	int	Toggles	size > 1
XVF_TOGGLE_TYPE	int	Toggles	KUIS_FLAG KUIS_LOGICAL KUIS_INTEGER KUIS_FLOAT KUIS_DOUBLE KUIS_STRING KUIS_INPUTFILE KUIS_OUTPUTFILE
XVF_TOGGLE_VAL	char *	Toggles	The current value of the toggle stored in a string,

E.10. Attributes of Lists

The following is a listing of GUI item attributes that apply to List, DisplayList, and StringList selections.

Descriptions of Attributes For List Selections	
Attribute	**Description**
XVF_LIST_ADD	This is a *write-only* attribute, ie, it can only be used with *xvf_set_attribute(s)()*. The label provided will be added to the end of the list: the new member will be assigned the next consecutive integer value following the value of the list member that was previously at the end of the list, and the size of the list incremented by 1. The list will be automatically updated to display the new label.
XVF_LIST_CONTENTS	This is the array of labels that is associated with each of the different values of the list. IMPORTANT NOTE: You MUST set XVF_LIST_SIZE to the number of elements in the list BEFORE setting XVF_LIST_CONTENTS. Neglecting to set XVF_LIST_SIZE to the number of elements in the list will result in memory corruption.
XVF_LIST_DELETE	This is a *write-only* attribute, ie, it can only be used with *xvf_set_attribute(s)()*. The list will be searched for the label provided, and that label will be deleted from the list, the size of the list decremented by 1. The list will be automatically updated to display the list without the deleted label. Because the integer values associated with lists are always incremental, if an item in the middle of the list is deleted, all items later in the list will have their values "scooted back" by 1. If the label specified cannot be found, the list will remain unaffected.

Descriptions of Attributes For List Selections	
Attribute	**Description**
XVF_LIST_DELETEALL	This is an *action* attribute, ie, it always takes a value of TRUE, and can only be used with *xvf_set_attribute(s)()*. All items will be deleted from the list, the size of the list will be set to 0. The list will be updated, and displayed as empty.
XVF_LIST_INDEX	This attribute is the index of the current value of the list. Suppose we have a list with values ranging from 15 to 24; that is, there are 10 items in the list, and the ListStart attribute is set to 10. We may have an array of strings defining the choices that correspond to the values 15 to 24, but if so, the indices into that array will be 0 to 9. This attribute stores the appropriate index. This value can also be determined using the equation: list_index = list_value - list_start
XVF_LIST_LABEL	This attribute is the label which corresponds to the *current value* of the list. Suppose we have a pull-down list with values ranging from 10 to 12, with associated labels "discrete", "bargraph", and "polymarker". If the current value of the list was 12, then the corresponding label will be "polymarker".
XVF_LIST_SIZE	This attribute specifies the number of items in the list. Note that this attribute MUST be set correctly before the contents of the list may be set or retrieved.
XVF_LIST_START	This attribute allows the incremental values of the list to begin at any integer. We may have a list with 10 items, but if the values of the list are to range from 15 to 24, the list start attribute will have a value of 15. NOTE: StringList selections do not have the ListStart attribute; the list start for a StringList selection is always 1.
XVF_LIST_VAL	This attribute is the integer value represented by the currently selected item from the list. Suppose we have a list with values ranging from 15 to 24, and the fifth item is selected; in this case, the integer value 19 will be the list value.

Summary of Attributes For List Selections			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_LIST_ADD	char *	Lists DisplayLists StringLists	any string to add as new list member
XVF_LIST_CONTENTS	char **	Lists DisplayLists StringLists	array of strings to establish as list members
XVF_LIST_DELETE	char *	Lists DisplayLists StringLists	string representing list member to delete

Summary of Attributes For List Selections			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_LIST_DELETEALL	int	Lists DisplayLists StringLists	TRUE (action attribute)
XVF_LIST_INDEX	int	Lists DisplayLists StringLists	XVF_LIST_START <= index < XVF_LIST_NUM+ XVF_LIST_START
XVF_LIST_LABEL	char *	Lists DisplayLists StringLists	any string
XVF_LIST_SIZE	int	Lists DisplayLists StringLists	size >= 0
XVF_LIST_START	int	Lists DisplayLists	any integer value
XVF_LIST_VAL	int	Lists DisplayLists StringLists	XVF_LIST_START < val <= XVF_LIST_START+ XVF_LIST_NUM

E.11. Attributes of Cycles

The following is a listing of GUI item attributes that apply to Cycle selections.

Descriptions of Attributes For Cycle Selections	
Attribute	**Description**
XVF_CYCLE_ADD	This is a *write-only* attribute, ie, it can only be used with *xvf_set_attribute(s)()*. The label provided will be added to the end of the cycle: the new member will be assigned the next consecutive integer value following the value of the cycle member that was previously at the end of the cycle, and the size of the cycle incremented by 1.
XVF_CYCLE_CONTENTS	This is the array of labels that is associated with each of the different values of the cycle. IMPORTANT NOTE: You MUST set XVF_CYCLE_SIZE to the number of elements in the cycle BEFORE setting XVF_CYCLE_CONTENTS. Neglecting to set XVF_CYCLE_SIZE to the number of elements in the cycle will result in memory corruption.

Descriptions of Attributes For Cycle Selections	
Attribute	**Description**
XVF_CYCLE_DELETE	This is a *write-only* attribute, ie, it can only be used with *xvf_set_attribute(s)()*. The cycle will be searched for the label provided, and that label will be deleted from the cycle, the size of the cycle decremented by 1. The cycle will be automatically updated to display the cycle without the deleted label. Because the integer values associated with cycles are always incremental, if an item in the middle of the cycle is deleted, all items later in the cycle will have their values "scooted back" by 1. If the label specified cannot be found, the cycle will remain unaffected.
XVF_CYCLE_DELETEALL	This is an *action* attribute, ie, it always takes a value of TRUE, and can only be used with *xvf_set_attribute(s)()*. All items will be deleted from the cycle, the size of the cycle will be set to 0. The cycle will be updated, and displayed as empty.
XVF_CYCLE_INDEX	This attribute is the index of the current value of the cycle. Suppose we have a cycle with values ranging from 10 to 14; that is, there are 5 items in the cycle, and the CycleStart attribute is set to 10. We may have an array of strings defining the choices that correspond to the values 10 to 14, but the indices into that array will be 0 to 4. This attribute stores the appropriate index. This value can also be determined using the equation: cycle_index = cycle_value - cycle_start
XVF_CYCLE_LABEL	This attribute is the label which corresponds to the *current value* of the cycle. Suppose we have a pull-down cycle with values ranging from 100 to 102, with associated labels "vegetation", "industrial", and "water". If the current value of the cycle was 102, then the corresponding label will be "water".
XVF_CYCLE_SIZE	This attribute specifies the number of items in the cycle. Note that this attribute MUST be set correctly before the contents of the cycle may be set or retrieved.
XVF_CYCLE_START	This attribute allows the incremental values of the cycle to begin at any integer. We may have a cycle with 10 items, but if the values of the cycle are to range from 25 to 34, the cycle start attribute will have a value of 25.
XVF_CYCLE_VAL	This attribute is the integer value represented by the currently selected item from the cycle. Suppose we have a cycle with values ranging from 15 to 24, and the fifth item is selected; in this case, the integer value 19 will be the cycle value.

Summary of Attributes For Cycle Selections			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_CYCLE_ADD	char *	Cycles	any string to add as new cycle member

Summary of Attributes For Cycle Selections			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_CYCLE_CONTENTS	char **	Cycles	array of strings to establish as cycle members
XVF_CYCLE_DELETE	char *	Cycles	string representing cycle member to delete
XVF_CYCLE_DELETEALL	int	Cycles	TRUE (action attribute)
XVF_CYCLE_INDEX	int	Cycles	XVF_CYCLE_START <= index < XVF_CYCLE_NUM+ XVF_CYCLE_START
XVF_CYCLE_LABEL	char *	Cycles	any string
XVF_CYCLE_SIZE	int	Cycles	size >= 0
XVF_CYCLE_START	int	Cycles	any integer value
XVF_CYCLE_VAL	int	Cycles	XVF_CYCLE_START < val <= XVF_CYCLE_START+ XVF_CYCLE_NUM

E.12. Attributes of Routine Buttons And Help Buttons

The following is a listing of GUI item attributes that only apply to Routine buttons and Help buttons.

Descriptions of Attributes For Miscellaneous Selections	
Attribute	**Description**
XVF_HELPPATH	This path may be the path to a specific help file to be displayed when the user clicks on the help button, or it may be the path to a directory in which two or more help files are contained. If the help path accesses a particular file, the help object will come up with that file displayed in it. If it accesses a directory, the help object that comes up will have an "Options" button which produces a pulldown menu at the upper left hand corner. There will be one entry in the pulldown menu for each file in the directory. When the user selects an item from the menu, the file that it names will be displayed. NOTE: if a directory is specified as the help path, it is important to make sure that no extraneous files exist in that directory, as the help object will create an item in the pulldown menu with the name of each file in the directory, and allow the user to access every file in the directory. NOTE: if the help path specified is invalid, the user will receive an error message when they click on the help button.
XVF_ROUTINE	This is the program that will be executed when the user clicks on the Routine button. It should be specified using the name of the toolbox bin in which the program binary is located and the name of the binary itself, using the syntax ${TOOLBOXNAME}BIN/{binary name}. For example, the karith1 program of the $DATAMANIP toolbox has its Routine attribute specified as $DATAMANIPBIN/karith1.

Summary of Attributes For Miscellaneous Selections			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_HELPPATH	char *	Help buttons	valid path to help page
XVF_ROUTINE	char *	Routine buttons	name of program to execute

E.13. Attributes for Subform And Pane Display

The following is a listing of GUI item attributes that are used for changing the subform and/or pane that is currently displayed.

Descriptions of Attributes For Subform & Pane Display	
Attribute	**Description**
XVF_DISPLAY_PANE	This attribute allows you to select or unselect a particular pane of a subform; it is only applicable when multiple panes exist on the subform. This is a *write-only* attribute; that is, it can only be used with *xvf_set_attribute(s)()*. Provide a value of TRUE (1) to select the specified pane, a value of FALSE (0) to unselect the specified pane. Requests to select panes that are already selected and requests to unselect panes that are not currently selected will have no effect.
XVF_DISPLAY_SUBFORM	This attribute allows you to unmap the currently displayed subform, or to to display or a currently unmapped subform. Accordingly, this attribute is only applicable to GUI's having master forms with multiple subforms. It is a *write-only* attribute; that is, it can only be used with *xvf_set_attribute(s)()*. Provide a value of TRUE (1) to map the desired subform, a value of FALSE (0) to unmap the desired subform. Requests to map subforms that are already mapped and requests to unmap subforms that are already unmapped will have no effect. If a master form has subforms that are MutuallyExclusive, such that only one subform may be mapped at any one time, a request to map a subform will cause the currently mapped subform to be popped down.

Summary of Attributes For Subform & Pane Display			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_DISPLAY_PANE	int	Panes	TRUE/FALSE
XVF_DISPLAY_SUBFORM	int	Subforms	TRUE/FALSE

E.14. Attributes for Printing UIS files

The following is a listing of GUI item attributes that are used for printing UIS files.

Descriptions of Attributes For Printing UIS's	
Attribute	**Description**
XVF_PRINT_PANE	This is a *write-only* attribute; that is, it can only be used with *xvf_set_attribute(s)()*. It causes the UIS file describing *only the specified pane* to be written out to the filename specified. If any changes have been made to the pane by the user via the menuforms, or any changes have been made to the pane by the application program via calls to *xvf_set_attribute(s)()*, those changes will be reflected in the resulting UIS file.
XVF_PRINT_SUBFORM	This is a *write-only* attribute; that is, it can only be used with *xvf_set_attribute(s)()*. It causes the UIS file describing *only the specified subform* to be written out to the filename specified. If any changes have been made to the subform by the user via the menuforms, or any changes have been made to the subform by the application program via calls to *xvf_set_attribute(s)()*, those changes will be reflected in the resulting UIS file.
XVF_PRINT_UIS	This is a *write-only* attribute; that is, it can only be used with *xvf_set_attribute(s)()*. It causes the UIS file describing the entire GUI exactly as it is displayed to be written out to the filename specified. If any changes have been made to the GUI by the user via the menuforms, or any changes have been made to the GUI by the application program via calls to *xvf_set_attribute(s)()*, those changes will be reflected in the resulting UIS file.

Summary of Attributes For Printing UIS's			
Attribute	**Type**	**GUI Items**	**Legal Values**
XVF_PRINT_PANE	char *	Panes	Name of file to which to print UIS describing pane
XVF_PRINT_SUBFORM	char *	Subforms	Name of file to which to print UIS describing subform
XVF_PRINT_UIS	char *	Forms	Name of file to which to print UIS describing entire GUI

E.15. xvf_get_xvobject() — *return desired xvobject component of kformstruct*

Synopsis

```
xvobject xvf_get_xvobject(
    kform_struct *kformstruct,
    int          item_part,
```

```
int          create)
```

Input Arguments

`kformstruct`

Pointer to the generic structure representing the GUI item's node in the form tree; appropriate candidates are automatically generated in "form_info.h" by conductor.

`item_part`

Indicates the part of the gui desired.

`create`

the xvobject in question may not have been created yet (ie, if a subform or a pane has not been mapped yet). Passing FALSE indicates that NULL should be returned for the xvobject if it has not yet been created; passing TRUE specifies that the object should be created and then returned.

Returns

The xvobject associated with the kformstruct on success. If 'create' is TRUE, NULL is returned only on failure; If 'create' is FALSE, NULL is returned when the xvobject has not yet been created.

Description

Given a pointer to a kformstruct, and an "item_part" indicator, returns the desired xvobject associated with the selection represented by the kformstruct, so that the application may make changes to its GUI items using xvw_set_attributes() which may not be supported by xvf_set_attributes().

NOTE: Please see xvf_set_attributes() before using this more difficult method of implementing GUI changes.

Depending on the type of GUI item, it may be made up of one or many xvobjects. For example, an action button consists only of a button GUI object. A Float selection, however, will have at least a backplane, a label object, and a text object. If it is optional, it will also have a small button object that serves as the optional box; if it is "live", it will have a pixmap object to display the stylized <cr> sign, and it may also contain a scrollbar object. The "item_part" argument is used to specify which part of the GUI item is desired.

For example, when passed a kformstruct associated with an Action Button and an "item_part" of XVF_BUTTON_OBJ, xvf_get_xvobject() will return the xvobject which is the GUI button object. When passed a kformstruct associated with an InputFile selection, and an "item_part" of XVF_BACKPLANE, xvf_get_xvobject() will return the xvobject which serves as the backplane for the InputFile selection. If an "item_part" is requested which does not exist in the GUI item represented by the kformstruct, an error message will be printed and NULL will be returned.

Legal Item Parts Include:

XVF_BACKPLANE : The backplane of the GUI item.
XVF_OPT_OBJ : The optional box of a GUI item (optional selection only).
XVF_BUTTON_OBJ : The button on a GUI item (GUI buttons, cycles, lists, etc)
XVF_LABEL_OBJ : The label on a GUI item

XVF_TEXT_OBJ : The text box of a GUI selection
XVF_SCROLL_OBJ : Scrollbar of a Float, Integer, or Double selection
XVF_PIXMAP_OBJ : Stylized <cr> pixmap of a "live" selection

This function allows the application to obtain the actual GUI objects from which GUI items are constructed. By setting attributes directly on the xvobjects themselves, the application may make certain changes to the physical appearance of its GUI during runtime, where those changes are not already supported by *xvf_set_attribute(s)()*. IMPORTANT NOTE: Please see the documentation on *xvf_set_attribute(s)()* first, before attempting to make changes directly to the GUI objects that comprise GUI items.

- Changing the border width of one of the GUI objects that make up the GUI item.
- Controlling the colors and/or fonts of used by GUI items from within the application.

For these and other reasons, the application programmer may want to call *xvf_get_xvobject()* in order to obtain one or more of the GUI objects that make up a GUI item. Then, *xvw_set_attribute(s)()* may be used directly on the GUI object to set attributes of the GUI object, and thus force them to appear differently than the *xvforms* library would otherwise display them.

All candidates for the *kformstruct* first parameter to *xvf_get_xvobject()* are defined for you by the xvroutine code generator as elements of the GUI Information structure which is generated in "form_info.h" (please see Chapter 4 of the VisiQuest 2001 Toolbox Programmer's Manual for more details). Pass the field in the GUI Information structure named "*var*_struct" where *var* is the variable name you provided on the UIS line that defines the GUI item for which you wish to obtain the xvobject building block(s).

Item Part	Item Part Description	Valid GUI Items
XVF_BACKPLANE	The backplane of a GUI item	Any GUI item. GUI items that are made up only of buttons (ie, Action Buttons, Routine Buttons, Quit Buttons, Help Buttons, etc) will return the GUI button object as the "backplane."
XVF_OPT_OBJ	The optional box of a GUI selection.	Optional selections ONLY, where selections may be of type InputFile, OutputFile, Integer, Float, Double, String, Logical, Cycle, List, StringList, or Toggle.
XVF_LABEL_OBJ	The label object of a GUI item.	All GUI items that have a separate label GUI object; all GUI items except Action Buttons, Help Buttons, Quit Buttons, Routine Buttons, Guide Buttons, and Subform Buttons.
XVF_BUTTON_OBJ	The button object of a GUI item.	All GUI items that are made up of a single GUI button object, or contain a separate button GUI object. The former include Action Buttons, Help Buttons, Quit Buttons, Routine Buttons, Guide Buttons, and Subform Buttons (note that passing XVF_BACKPLANE for these will achieve the same purpose); the latter include Cycle, List, Logical, and StringList selections.
XVF_TEXT_OBJ	The text object of a GUI item.	Only GUI selections which contain text GUI objects, which include: InputFile, OutputFile, Integer, Float, Double, String, and StringList.

F. Adding Extra Calls To GUI Items

F.1. xvf_add_extra_call() — *add extra callback to GUI item*

Synopsis

```
int xvf_add_extra_call(
   kform_struct *kformstruct,
   void        (*routine)(kaddr),
   kaddr        client_data,
   int          call_location)
```

Input Arguments

`kformstruct`

kform_struct associated with the GUI item, to which the extra call is to be added. Note that all candidates for this argument will be found in the GUI Information structure, which is automatically generated in the "form_info.h" file.

`routine`

The extra routine to call. This routine MUST be declared as follows:

```
void routine(
    kaddr client_data)
```

`client_data`

Client data with which to call extra routine. The client_data pointer is used when it is necessary to pass arguments to a callback routine. First, define a single structure containing all the parameters needed. Declare a pointer to the structure, dynamically allocate the pointer, and initialize all the fields with the values that you would have given to the parameters before passing them to the routine in a normal call. Pass the pointer to your allocated, initialized structure to xvf_add_extra_call() as the client_data. Inside the callback, cast the client_data to the appropriate structure type, as in:

```
my_struct *my_ptr = (my_struct *) client_data;
```

At that point, the fields may be accessed as desired. They will hold the values to which they were initialized.

`call_location`

tells whether the specified routine should be called before the "normal" operation of the item is performed, after the "normal" operation of the item is performed, or instead of the "normal" operation of the item.

One of: XVF_CALL_FIRST, XVF_CALL_LAST, or XVF_CALL_SUBSTITUTE

Returns

TRUE on success, FALSE on failure

Description

Some items on the VisiQuest GUI perform a particular function automatically, *without ever returning flow control to the application program*; they cannot be made "live" in order to force a return of flow control to the application. These items include:

(1) subform buttons, which map a subform
(2) guide buttons, which map a pane
(3) quit buttons, which close the subform or exit the program
(4) help buttons, which put up a help page

With these items, a mechanism is sometimes desired which

will allow the programmer to call a particular routine when these items are used. For example, the programmer may want a certain routine to be called before a subform is mapped, after a pane is mapped, before the GUI is destroyed, or before a help page is displayed. For situations like this, xvf_add_extra_call() may be used to specify an additional routine that will be called by the built-in callback for the GUI item in question. xvf_add_extra_call() may be called more than once, if calls to more than one extra routine are needed.

Restrictions

Selection types other than buttons (ie, input files, output files, integers, floats, doubles, cycles, logicals, strings, stringlists, displaylists, etc.) may only have extra callbacks added with XVF_CALL_LAST.

F.2. xvf_remove_extra_call() — *remove function call from GUI item*

Synopsis

```
int xvf_remove_extra_call(
   kform_struct *kformstruct,
   void         (*routine)(kaddr),
   kaddr         client_data)
```

Input Arguments

`kformstruct`

kform_struct associated with the GUI item, from which the extra call is to be removed (passed to xvf_add_extra_call earlier).

`routine`

The extra routine that was being called (passed to xvf_add_extra_call earlier).

`client_data`

Client data with which routine was called (passed to xvf_add_extra_call earlier).

Returns

TRUE on success, FALSE on failure

Description

Removes an extra call from a subform button, guide button, quit button, help button, answer infile selection, or answer outfile selection, where the extra call was added earlier with xvf_add_extra_call().

Restrictions

This routine should only be used for extra calls that were added earlier with a call to xvf_add_extra_call.

G. Adding Callbacks To GUI Attributes

G.1. xvf_add_gui_callback() — *add callback to a GUI item*

Synopsis

```
int xvf_add_gui_callback(
   kform_struct *kformstruct,
   char         *attribute,
   void         (*routine)(kform_struct *, char *, kaddr, kaddr),
   kaddr        client_data)
```

Input Arguments

`kformstruct`

kform_struct associated with the GUI item, to which the callback is to be added. Note that all candidates for this argument will be found in the GUI Information structure, which is automatically generated in the "form_info.h" file.

`attribute`

the xvforms attribute on which to fire the callback. For all GUI item attribute changes, provide, "xvf_all_attributes".

`routine`

The callback routine to fire. This routine MUST be declared as follows:

```
void routine(kform_struct kformstuct,
        char  *attribute,
        kaddr attr_value,
        kaddr client_data)
```

`client_data`

Client data with which to call extra routine. The client_data pointer is used when it is necessary to pass arguments to a callback routine. First, define a single structure containing all the parameters needed. Declare a pointer to the structure, dynamically allocate the pointer, and initialize all the fields with the values that you would have given to the parameters before passing them to the routine in a normal call. Pass the pointer to your allocated, initialized structure to xvf_add_extra_call() as the client_data. Inside the callback, cast the client_data to the appropriate structure type, as in:

my_struct *my_ptr = (my_struct *) client_data;

At that point, the fields may be accessed as desired. They will hold the values to which they were initialized.

Returns

TRUE on success, FALSE on failure

Description

This function adds a callback to a GUI item that will be called if an attribute of the GUI item itself is changed.

G.2. xvf_remove_gui_callback() — *remove callback from GUI item*

Synopsis

```
int xvf_remove_gui_callback(
   kform_struct *kformstruct,
   char         *attribute,
   void         (*routine)(kform_struct *, char *, kaddr, kaddr),
   kaddr        client_data)
```

Input Arguments

`kformstruct`

kform_struct associated with the GUI item, from which the callback is to be removed (passed to xvf_add_gui_callback earlier).

`attribute`

the xvforms attribute on which the callback was fired. For all GUI item attribute changes, provide, "xvf_all_attributes".

`routine`

The extra routine that was being called (passed to xvf_add_gui_callback earlier).

`client_data`

Client data with which routine was called (passed to xvf_add_gui_callback earlier).

Returns

TRUE on success, FALSE on failure

Description

This function removes a callback from a GUI item that was being called when an attribute of the GUI item was changed.

Restrictions

This routine should only be used for callbacks that were added earlier to GUI items with a call to xvf_add_gui_callback.

This page left intentionally blank

Table of Contents

Chapter 9

Xvutils

Chapter 9 - Xvutils

A. Introduction

The *xvutils* library contains utility functions that are used to augment the Graphical User Interface (GUI) created with the *xvforms* library. These utilities generally involve pop-up I/O display objects, such as file browsers, list objects, pop-up error messages, and text display objects.

B. Errors, Warnings, Prompts, and Information Display

The most commonly-used utilities in the *xvutils* library are *not* called directly from the application. Direct calls to these routines violate the VisiQuest 2001 programming standard. They are included here in Chapter 7, *xvutils* Library to emphasize the fact that standardized error reporting, warning messages, choice prompts, and information display are available for xvroutines and can be utilized. These routines are:

• *xvu_choose_wait()* - pop up a choose dialog box; wait for response
• *xvu_error_wait()* - pop up error object (1 button); wait for acknowledgement
• *xvu_info_wait()* - pop up info object (1 button); wait for acknowledgement
• *xvu_multiprompt_wait()* - pop up dialog containing prompt and acknowledgement buttons
• *xvu_prompt_wait()* - pop up prompting object (2 choices); wait for response
• *xvu_quit_wait()* - pop up a quit dialog box; wait for response
• *xvu_save_wait()* - pop up a save message; wait for response
• *xvu_warn_wait()* - pop up warn object (1 button); wait for acknowledgement

Instead of calling these *xvutils* routines from the application, it is recommended that xvroutines (like kroutines and hybrids) make calls to the appropriate *kutils* information/error reporting facilities. In the same order as their *xvutils* counterparts. These are:

• *kerror()* - print error messages in a standardized format
• *kinfo()* - print information messages in a standardized format
• *kprompt()* - request an acknowledgement from the user
• *kchoose()* - prompt the user to select from a list of items
• *ksave()* - request an acknowledgement for quitting an application
• *kquit()* - request an acknowledgement for quitting an application

When the display is open (i.e., GUI is displayed) these *kutils* routines call the appropriate *xvutils* utility; the error message, warning message, choice prompt, info message, save message, or quit message pops-up on the display and the application is *blocked until the user provides a response*.

However, if the display is *not* open, the *kutils* routines call their own default utilities, and the message and/or prompt in question will correctly appear in the user's *tty*. By contrast, if an application calls one of the *xvutils* utilities directly and the routine call is encountered when the display is not open, the message will be displayed, but the user cannot be prompted for a response. In the case of warnings and choice prompts where acknowledgement is necessary, this will cause inappropriate behavior on the part of the application.

For those who want to cover the cases of both open and closed displays, the *kutils* routines are repeated here (they are also listed under the chapter for *kutils*):

B.1. kerror() — *print error messages in a standardized format*

Synopsis

```
int kerror(
   char *library,
   char *routine,
   char *format,
   kvalist)
```

Input Arguments

`library`
: name of library (NULL if not applicable)

`routine`
: name of routine

`format`
: grammatically correct, clear explanation of the error that occurred. This can be formatted like a printf statement

Returns

TRUE if the error was successfully acknowledge, otherwise if the message was not acknowledged FALSE is returned.

Description

kerror produces standardized error messages for VisiQuest library routines and applications. It should be called in EVERY instance of error messaging by EVERY VisiQuest function, subroutine, or main program. If library is NULL then the program name will be used as the source of the error.

B.2. kinfo() — *print information messages in a standardized format*

Synopsis

```
int kinfo(
   int notify_type,
   char *format,
   kvalist)
```

Input Arguments

`notify_type`
: the notify level specified by the programmer KFORCE, KSTANDARD, KVERBOSE

`format`
grammatically correct, clear explanation of the information. Note the message can be formatted like a printf statement.

Returns

TRUE if the message was successfully printed, otherwise FALSE is returned.

Description

kinfo produces standardized information messages for VisiQuest library routines and VisiQuest applications. It should be called in EVERY instance of information messaging by EVERY VisiQuest function, subroutine, or main program.

If the notify_type variable is set to KFORCE, then the prompt will always appear regardless of the setting of the environment variable KHOROS_NOTIFY.

If the notify_type variable is set to KSTANDARD and the user has the environment variable KHOROS_NOTIFY set to either STANDARD or VERBOSE the prompt will appear.

And finally, if the notify_type variable is set to KVERBOSE and the environment variable KHOROS_NOTIFY set to VERBOSE, the prompt will appear.

Here is a summary table:

```
notify_type = FORCE     always prompt, ignore the setting of
                        the environment variable KHOROS_NOTIFY

notify_type = STANDARD  only prompt when the environment
                        variable KHOROS_NOTIFY is set to
                        STANDARD or VERBOSE.

notify_type = VERBOSE   only prompt when the environment
                        variable KHOROS_NOTIFY is set to
                        VERBOSE
```

B.3. kprompt() — *request an acknowledgement from the user*

Synopsis

```
int kprompt(
   int notify_type,
```

```
char *yes_response,
char *no_response,
int default_val,
char *format,
kvalist)
```

Input Arguments

`notify_type`
: the notify level specified by the programmer KFORCE, KSTANDARD, KVERBOSSE

`yes_response`
: name of "yes" response string ("Yes" if NULL)

`no_response`
: name of "no" response string ("No" if NULL)

`default_val`
: the default value to list when prompting

`format`
: grammatically correct, clear explanation of the error that occurred. This can be formatted like a printf statement.

Returns

TRUE if the prompt was successfully acknowledged, otherwise if the message was not acknowledged FALSE is returned. In the event and error occurs the default value is returned.

Description

kprompt will call the specified prompt handler to request or demand an acknowledgement from the user. This utility can operate in several different modes.

If the notify_type variable is set to KFORCE, then the prompt will always appear regardless of the setting of the environment variable KHOROS_NOTIFY.

If the notify_type variable is set to KSTANDARD and the user has the environment variable KHOROS_NOTIFY set to either STANDARD or VERBOSE the prompt will appear.

And finally, if the notify_type variable is set to KVERBOSE and the environment variable KHOROS_NOTIFY set to VERBOSE, the prompt will appear.

Here is a summary table:

```
notify_type = FORCE     always prompt, ignore the setting of
                        the environment variable KHOROS_NOTIFY

notify_type = STANDARD  only prompt when the environment
                        variable KHOROS_NOTIFY is set to
                        STANDARD or VERBOSE.

notify_type = VERBOSE   only prompt when the environment
```

```
variable KHOROS_NOTIFY is set to
VERBOSE
```

B.4. kchoose() — *prompt the user to select from a list of items*

Synopsis

```
char *kchoose(
   int notify_type,
   char **list_of_options,
   int num_options,
   int default_index,
   char *return_string,
   int *return_index,
   char *format,
   kvalist)
```

Input Arguments

`notify_type`
: the notify level specified by the programmer KFORCE, KSTANDARD, KVERBOSSE

`list_of_options`
: an array of strings containing the items to select from.

`num_options`
: The number of items in the list_of_options.

`default_index`
: The index number to the default item, must start at 1.

`format`
: grammatically correct, clear explanation of the error that occurred. This can be formatted like a (void) printf statement.

Output Arguments

`return_string`
: string that holds the selected item. If it's NULL, it kmallocs the space necessary, and returns the string.

`return_index`
: This is the index of the item selected.

Returns

return_string if it is not NULL, or a pointer to the resulting kmalloc'ed string if it is NULL. NULL is returned upon error.

Description

kchoose will call the specified choose handler to request the user to make a selection from a list of items. This utility can operate in several different modes.

If the notify_type variable is set to KFORCE, then the prompt will always appear regardless of the setting of the environment variable KHOROS_NOTIFY.

If the notify_type variable is set to KSTANDARD and the user has the environment variable KHOROS_NOTIFY set to either KSTANDARD or KVERBOSE the prompt will appear.

And finally, if the notify_type variable is set to KVERBOSE and the environment variable KHOROS_NOTIFY set to KVERBOSE, the prompt will appear.

Here is a summary table:

```
notify_type = FORCE     always prompt, ignore the setting of
                        the environment variable KHOROS_NOTIFY

notify_type = STANDARD  only prompt when the environment
                        variable KHOROS_NOTIFY is set to
                        STANDARD or VERBOSE.

notify_type = VERBOSE   only prompt when the environment
                        variable KHOROS_NOTIFY is set to
```

B.5. ksave() — *request an acknowledgement for quitting an application*

Synopsis

```
int ksave(
   int notify_type,
   char *format,
   kvalist)
```

Input Arguments

`notify_type`

the notify level specified by the programmer KFORCE, KSTANDARD, KVERBOSE message - the message prompting the user for exiting.

Returns

TRUE if the user wants to quit the application, otherwise if the user doesn't want to quit then FALSE

is returned. In the event of an error FALSE is returned.

Description

ksave will call the specified prompt handler to request or demand an acknowledgement from the user. This utility will as the user if it is ok to overwrite the file in question, and can operate in several different modes.

If the notify_type variable is set to KFORCE, then the prompt will always appear regardless of the setting of the environment variable KHOROS_NOTIFY.

If the notify_type variable is set to KSTANDARD and the user has the environment variable KHOROS_NOTIFY set to either STANDARD or VERBOSE the prompt will appear.

And finally, if the notify_type variable is set to KVERBOSE and the environment variable KHOROS_NOTIFY set to VERBOSE, the prompt will appear.

Here is a summary table:

```
notify_type = FORCE     never prompt when the environment
                        variable KHOROS_NOTIFY is set to
                        FORCE.

notify_type = STANDARD  only prompt when the environment
                        variable KHOROS_NOTIFY is set to
                        STANDARD or VERBOSE.

notify_type = VERBOSE   only prompt when the environment
                        variable KHOROS_NOTIFY is set to
                        VERBOSE
```

B.6. kquit() — *request an acknowledgement for quitting an application*

Synopsis

```
int kquit(
   int notify_type,
   char *format,
   kvalist)
```

Input Arguments

```
notify_type
```

the notify level specified by the programmer KFORCE, KSTANDARD, KVERBOSE message - the message prompting the user for exiting.

Returns

TRUE if the user wants to quit the application, otherwise if the user doesn't want to quit then FALSE is returned. In the event of an error FALSE is returned.

Description

kquit will call the specified prompt handler to request or demand an acknowledgement from the user. This utility will as the user if it is ok to overwrite the file in question, and can operate in several different modes.

If the notify_type variable is set to KFORCE, then the prompt will always appear regardless of the setting of the environment variable KHOROS_NOTIFY.

If the notify_type variable is set to KSTANDARD and the user has the environment variable KHOROS_NOTIFY set to either STANDARD or VERBOSE the prompt will appear.

And finally, if the notify_type variable is set to KVERBOSE and the environment variable KHOROS_NOTIFY set to VERBOSE, the prompt will appear.

Here is a summary table:

```
notify_type = FORCE     never prompt when the environment
                        variable KHOROS_NOTIFY is set to
                        FORCE.

notify_type = STANDARD  only prompt when the environment
                        variable KHOROS_NOTIFY is set to
                        STANDARD or VERBOSE.

notify_type = VERBOSE   only prompt when the environment
                        variable KHOROS_NOTIFY is set to
                        VERBOSE
```

For those uninterested in the closed display or don't care to follow the VisiQuest 2001 programming standard, their function definitions are as follows:

B.7. xvu_choose_wait() — *pop up a choose dialog box; wait for response*

Synopsis

```
int xvu_choose_wait(
    char  *prompt,
    char  **item_labels,
    int   item_num,
    int   default_val,
    int   user_defined,
    char  **return_string)
```

Input Arguments

`prompt`
: prompt to display at top of list object

`item_labels`
: array of strings to be displayed as items in the list

`item_num`
: size of the 'list' array

`default_val`
: number of the item in the list to return (starting at 1 for the first item) if the user clicks on "Cancel". Specify default of - 1 if the default is to be NULL.

`user_defined`
: flag indicating if a string object should be created at the bottom of the list object to allow the user to enter a string that is not part of the list array. If user defined items are allowed, and the user specifies their own list item, the number returned will be -1, indicating that the user's choice was not a member of the original list.

`return_string`
: be sure to set this to NULL if you want the string allocated for you; be sure to allocate sufficient space for the string otherwise!

Output Arguments

`return_string`
: the string chosen from the list by the user

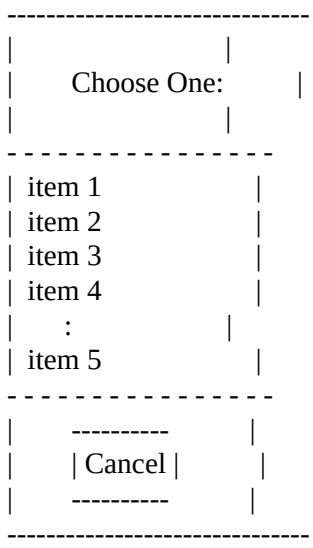
Returns

Value returned to caller will be: 1 if user selected item1 2 if user selected item2 3 if user selected item3 N if user selected itemN. 0 if there was an error creating selection object, or the user clicked on CANCEL and no default value (ie, a default value of -1) was provided. -1 if the user_defined flag was passed in as TRUE, and the user entered their own item to use.

This routine will NOT return to the calling program until one of the items is chosen from the list.

Description

Creates and maps a pop-up choose dialog box that consists of a list with N items, and looks like:



You may specify a prompt to replace "Choose 1". You may specify a label to replace "Cancel" on the button.

If desired, you may specify the "user defined" option to be TRUE, which will cause a text object to be created underneath the list. The user may then enter their own selection into the text object and hit <Enter>, which will cause the new selection to appear in the list, where it may be chosen.

IMPORTANT NOTE: to be consistent with the standards of the VisiQuest Software development system, you should really be calling kchoose(), not xvu_choose_wait().

B.8. xvu_error_wait() — *pop up error object (1 button); wait for acknowledgement*

Synopsis

```
int xvu_error_wait(
   char *error_mesg,
   char *error_label,
   char *button_label)
```

Input Arguments

`error_mesg`
: string describing error message

`error_label`
: short label for top of error object; passing NULL will result in default of "Error".

`button_label`
: label for acknowledgment button; passing NULL will result in default of "Ok".

Returns

Returns FALSE if it failed to create the error object otherwise waits for user to acknowledge error message,

Description

Creates a pop-up error object which must be acknowledged by the user before control is returned to the application program.

IMPORTANT NOTE: to be consistent with the standards of the VisiQuest Software development system, you should really be calling kerror(), not xvu_error_wait().

B.9. xvu_info_wait() — *pop up info object (1 button); wait for acknowledgement*

Synopsis

```
int xvu_info_wait(
   char *info_mesg,
   char *info_label,
   char *button_label)
```

Input Arguments

`info_mesg`
: string describing info message

`info_label`
: short label for top of info object; passing NULL will result in default of "Information".

`button_label`
: label for acknowledgment button; passing NULL will result in default of "Ok".

Returns

Returns FALSE if it failed to create the info object otherwise waits for user to acknowledge info message,

Description

Creates a pop-up info object which must be acknowledged by the user before control is returned to the application program.

IMPORTANT NOTE: to be consistent with the standards of the VisiQuest Software development system, you should really be calling kinfo(), not xvu_info_wait().

B.10. xvu_multiprompt_wait() — *pop up dialog containing prompt and acknowledgement buttons*

Synopsis

```
int xvu_multiprompt_wait(
   char *prompt,
   char *pixmapfile,
   int   num,
   char **labels,
   int  *values,
   int   def_action,
   int   num_across)
```

Input Arguments

`prompt`
: string that will appear as the prompt at the top of the dialog

`pixmapfile`
: if desired, the name of a pixmap to appear at the upper left hand corner may be specified; pass NULL for no pixmap.

`num`
: number of acknowledgement buttons desired

`labels`
: string array of desired labels for the buttonn (array must be of size specified by 'num'). For example, a string array containing the strings "Yes", "No" and "Cancel" would result in the popup dialog buttons being labeled "Yes", "No", and "Cancel", in that order.

`values`
: array of integers representing desired return values for the buttons (array must be of size specified by 'num'). In the example above, an array with values of [1,0,-1] would cause this routine to return 1 for "Yes", 0 for "No", and -1 for "Cancel".

`def_action`
: value of the button that will represent the default action. In the above example, if the default action would be to Cancel, then this value would be passed in as -1, since the value specified for "Cancel" is -1. When the dialog box pops up, it will be automatically positioned so that the pointer is over the "Cancel" button. If the user uses the window manager to pop down the dialog, a return value of -1 will be returned to the calling routine.

`num_across`
: number of buttons desired in each row across the bottom of the dialog

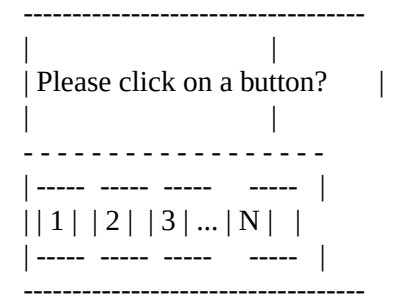
Returns

depending on the button selected, returns the corresponding integer value specified for that button in

the 'values' array

Description

Creates and maps a generalized pop-up dialog object which looks like:



This routine will block input to the application until it is acknowledged by the user. The user must click on one of the buttons before control will be returned to the application program; a return value indicates which button was chosen by the user.

B.11. xvu_prompt_wait() — *pop up prompting object (2 choices); wait for response*

Synopsis

```
int xvu_prompt_wait(
   char *prompt,
   char *label,
   char *yes_label,
   char *no_label,
   int  default_val)
```

Input Arguments

`prompt`
: prompting string

`label`
: short label for top of prompting object; NULL will produce default of "Choose One".

`yes_label`
: label for button representing value of 1 (TRUE); NULL will produce default of "Yes".

`no_label`
: label for second representing value of 0 (FALSE); NULL will produce default of "No".

`default_val`
: default value of 1 (TRUE) or 0 (FALSE)

Returns

1 (TRUE) if user selected button with yes_label (Yes) 0 (FALSE) if user selected button with no_label (No)

Description

Creates and maps a pop-up dialog object which has two buttons representing boolean values. The user must click on one of the buttons; a status representing which of the two buttons was chosen will be returned to the application. The user must click on one of the two buttons before control will be returned to the application program.

B.12. xvu_quit_wait() — *pop up a quit dialog box; wait for response*

Synopsis

```
int xvu_quit_wait(
   char *string,
   char *quit_label,
   char *cancel_label)
```

Input Arguments

`string`

Optional string for first line of label; if non-NULL, will replace "Please Confirm Exit\nExit program?"

`quit_label`

Optional label for button which returns value of 1; if non-NULL, will replace label of "Exit" on first button

`cancel_label`

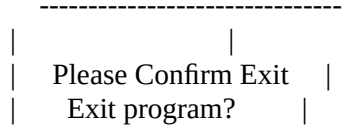
Optional label for button which returns value of 0; if non-NULL, will replace label of "Cancel" on first button

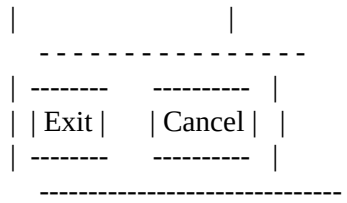
Returns

1 if user selected "Exit" (first) button 0 if user selected "Cancel" (second) button

Description

Creates and maps a pop-up quit dialog box which looks like:





The quit dialog box will block input to the application until it is acknowledged by the user. The user must click on "Exit" or "Cancel" before control will be returned to the application program.

B.13. xvu_save_wait() — *pop up a save message; wait for response*

Synopsis

```
int xvu_save_wait(
   char *string,
   char *filename,
   char *save_label,
   char *discard_label,
   char *cancel_label)
```

Input Arguments

`string`
Optional string for first line of label; if non-NULL, will replace "Save Changes to"

`filename`
Name of file to which changes will be saved (subsequently, by the application).

`save_label`
Optional label for button which returns value of 2; if non-NULL, will replace label of "Save" on first button

`discard_label`
Optional label for button which returns value of 1; if non-NULL, will replace label of "Discard" on first button

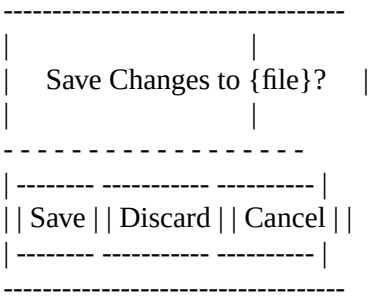
`cancel_label`
Optional label for button which returns value of 0; if non-NULL, will replace label of "Cancel" on first button

Returns

2 if user selected "Save" (first) button 1 if user selected "Discard" (second) 0 if user selected "Cancel" (third) button

Description

Creates and maps a pop-up save object which looks like:



This routine will block input to the application until it is acknowledged by the user. The user must click on "Save", "Discard" or "Cancel" before control will be returned to the application program.

Note that this utility does *not* actually save any changes to the file, or write to the file specified in any way. It simply prompts the user and returns a status depending on which button the user clicked.

B.14. xvu_warn_wait() — *pop up warn object (1 button); wait for acknowledgement*

Synopsis

```
int xvu_warn_wait(
   char *warn_mesg,
   char *warn_label,
   char *button_label)
```

Input Arguments

`warn_mesg`
: string describing warn message

`warn_label`
: short label for top of warn object; passing NULL will result in default of "Warning".

`button_label`
: label for acknowledgment button; passing NULL will result in default of "Ok".

Returns

Returns FALSE if it failed to create the warn object otherwise waits for user to acknowledge warn message,

Description

Creates a pop-up warn object which must be acknowledged by the user before control is returned to the application program.

IMPORTANT NOTE: to be consistent with the standards of the VisiQuest Software development system, you should really be calling kwarn(), not xvu_warn_wait().

C. Browser, Online Help, Misc Prompting, Lists, and File Viewing

The following routines pop-up a compound GUI I/O object, and will *block the application until the user acknowledges them* by selecting one of the buttons. These routines are as follows:

- *xvu_browse_wait()* - pop up the VisiQuest file/alias browser; wait for response
- *xvu_help_wait()* - display help file or help directory
- *xvu_query_wait()* - pop up prompt widget; wait for response
- *xvu_run_list_multsel_wait()* - display list, wait for multiple choices and acknowledgement

C.1. xvu_browse_wait() — *pop up the VisiQuest file/alias browser; wait for response*

Synopsis

```
char *xvu_browse_wait(
   char *directory)
```

Input Arguments

`directory`
path to initial directory

Returns

The string chosen from the browser by the user, or NULL if the user clicked on "Cancel".

Description

Causes the VisiQuest file / alias browser to pop up, and waits until the user selects a file or an alias before returning their selection.

At the top of the browser is a button with which the user may control whether the browser operates on files/directories or aliases. For more information on the Aliases capability, see Chapter 1 of the VisiQuest User's Manual.

The second element of the list is "../", which is used to move to an upper level directory. The remaining elements of the browser list are the contents of the specified directory. The browser remains displayed until the user selects a file from the browser, or clicks on "Cancel".

The user may select a file from the browser list; when this happens, the full path to the file is returned to the calling routine.

Alternatively, the user may select a sub-directory to change to that directory, or "../" to go up in the directory structure.

A text object at the bottom of the browser allows the user to type their own filename, or change directories. If a directory path is typed into the bottom text object followed by a <return>, files in the browser list will be updated with the contents of the new directory.

File/alias completion is supported; enter a few letters and use the Tab key to complete the entry.

C.2. xvu_help_wait() — *display help file or help directory*

Synopsis

```
int xvu_help_wait(
   char *path,
   char *label,
   int  x,
   int  y,
   int  width,
   int  height,
   int  interpret_roff)
```

Input Arguments

`path`
: the path of the (single) file to be displayed, or the directory containing the (multiple) files to be displayed in the help object. The path may include a '~' if desired, or may contain a reference to $TOOLBOX.

`label`
: short label for top of help object; NULL gives default of "Help".

x,
`y`
: (X, Y) location for GUI autoplacement of help window; use (-1, -1) for user placement of help window

`width`
: Width of window in pixels. If (-1) is specified, then a "good" size is provided automatically.

`height`
: Height of window in pixels. If (-1) is specified, then a "good" size is provided automatically.

`interpret_roff`
: Pass TRUE if files to be displayed will or may contain roff commands that will need to be formatted. Pass FALSE if files are to be displayed verbetim with no formatting.

Returns

TRUE on success, FALSE on failure

Description

Create a large read-only text object in which one or more ascii files may be displayed.

If the path specified is to a single file, that file is displayed. If the path specified is a directory, the text object will have one button at the top labeled with the name of each file in the directory. Clicking on such a button will cause the contents of the file labeled to appear in the text object. In either case, the on-line help object will be displayed until the user clicks on the "Quit" button. This utility will not grab events from the application program.

C.3. xvu_query_wait() — *pop up prompt widget; wait for response*

Synopsis

```
int  xvu_query_wait(
   char *top_label,
   char *prompts[],
   char *button,
   char *answers[],
   int  num_prompts,
   int  size)
```

Input Arguments

`top_label`
label to appear at top of query object prompts[] - array of prompts

`button`
label for acknowledgement button; NULL will give default of "Ok" answers[] - array of strings in which to return the user's responses, one for each of the prompts. Default values may be provided here if desired.

`num_prompts`
size of prompts[] and answers[] arrays

`size`
size (in characters) to make each text object for the users' responses

Returns

1 if user selected "Ok" 0 if user selected "Cancel"

Description

Creates a pop-up query object in order to prompt more responses from the user; xvu_query_wait may be used to obtain a set of strings, floats, integers, or responses of mixed types from the user.

Will not return control to application program until the user selects "Ok" or "Cancel".

The prompts[] and answers[] arrays must be of the same size, where that size is specified as 'num_prompts'.

The prompts[] array must be completely filled out with 'num_prompts' strings giving appropriate prompt for each response desired.

The answers[] array must be dynamically allocated, and each element of the string array should be initialized to contain the string representation of the default (if any). Strings should be passed in as NULL if no default is appropriate). The defaults will appear in the text objects when the query object is popped up; when the user clicks on "Ok", the contents of the answers[] array will be freed if non-NULL, and the responses of the user will be substituted in their place.

When the caller is prompting for float, int, or other non-string responses, it is expected that the caller

will convert the responses returned in the answers[] array to their appropriately typed counterparts before use.

The 'size' argument is provided to ensure that the text objects are physically large enough to accomodate the user's response.

Side Effects

The strings in the answers[] array will be freed if non-NULL, and the responses of the user substituted. If the user leaves the default in place, that will be returned; if the user blanks out a string, NULL is returned.

C.4. xvu_run_list_multsel_wait() — *display list, wait for multiple choices and acknowledgement*

Synopsis

```
xvw_list_struct **xvu_run_list_multsel_wait(
    char   *list[],
    size_t  size,
    char   *prompt,
    char   *label,
    int    user_defined,
    int    duplicates_ok,
    int    *num)
```

Input Arguments

`list`
: array of strings to be displayed in the list

`size`
: size of the 'list' array

`prompt`
: prompt to display the list object

`label`
: label to display at top of list object

`user_defined`
: flag indicating if a string object should be included to allow the user to enter a string that is not part of the list array.

`duplicates_ok`
: pass TRUE if the user is allowed to select the same item twice, FALSE otherwise

Output Arguments

`num`
: returns the number of items selected from the list.

Returns

An array of xvw_list_structs containing the strings and the indices of those strings describing the selections made from the list by the user. Returns NULL if the user clicks on "Cancel".

Description

Takes an array of strings, and uses them to create a pop-up list object. The user is allowed to select as many different strings as desired from the list; those strings are then returned to the caller with their indices in an array of xvu_list_struct's. If the user clicks on the "Cancel" button, NULL is returned.

Control is not returned to the application program until the user chooses an item from the list, or clicks on "cancel".

If the "duplicates_ok" flag is passed as FALSE, the user will only be allowed to select any given item from the list once. Selected items will be marked with a star. For example, the item "depth" would be displayed as "depth" if it had never been selected, but would be displayed as "* depth" after it was selected. A second click on the entry would cause it to be de-selected, and again it would be displayed as "depth". The array of xvu_list_struct's returned will have all unique elements; elements will appear according to the order in which selections were made by the user.

If the "duplicates_ok" flag is passed as TRUE, the user will be allowed to select any item from the list as many times as desired. Each selection from the list will cause that item to have the number in front of it incremented. For example, the item "depth" would be displayed as "depth" if it had never been selected, but "(1) depth" after the first time it was selected, and "(2) depth" after the second time it was selected. The array of xvu_list_struct's returned will have identical elements for each item that is selected multiple times. The elements of the xvu_list_struct array will appear in the order in which the items were selected by the user. For example, if the user chose "depth" twice, and then "width", and then "depth" again before clicking on "Ok", the array of xvu_list_structs returned would appear in the following order (identified by list_struct->string): depth, depth, width, depth.

If the "user_defined" flag is passed as TRUE, a string object will be created at the bottom of the list object where the user will be allowed to enter their own new value for the list. When the user hits <cr> following their newly defined list item, the new value will be added to the list, and may subsequently be selected, either once (if duplicates are not allowed), or many times (if duplicates are allowed).

Table of Contents

This page left intentionally blank

Chapter 10

Xvlang

Chapter 10 - Xvlang

A. Introduction

The *xvlang* visual programming toolkit supports an object oriented approach to the design and implementation of visual programming languages. The visual programming objects offered by *xvlang* toolkit are used by VisiQuest, the visual programming language of the VisiQuest 2001 system. VisiQuest itself is actually a relatively small application. Most of the functionality needed to implement it is provided by the visual programming objects of the *xvlang* library. Admittedly, the visual programming objects offered by the *xvlang* library are strongly influenced by the specific needs of VisiQuest; however, the visual programming toolkit is designed to be as general and as flexible as possible, in order to support creation of new applications that follow the visual programming paradigm.

Use of the visual programming objects available in *xvlang* allows flexibility and reusability to experiment with different visual programming applications, and offers the possibility of adapting existing models to meet the needs of new visual languages.

To create a new visual programming environment using the visual programming objects available in *xvlang*, there are only two major tasks involved: (1) to create a new visual programming object, subclassed from the Node object (see Section B.3), that enforces the desired model, and (2) to overload new save and restore methods in the Workspace object (see Section B.2) that will handle the visual network appropriately.

A visual language developed as a single monolithic application cannot easily be made flexible or expandable. By providing a visual programming object which addresses each key component of the visual programming environment, *xvlang* decouples the complexity of those visual programming components from the visual programming environment itself. This allows the developer of the visual programming language to concentrate on the visual programming paradigm to be designed, rather than on the functionality of the various components necessary to the language.

Available Functions

- *xvw_create_node()* - create an node object
- *xvw_create_glyph()* - create a glyph object
- *xvw_create_conditional()* - create a conditional object
- *xvw_create_procedure()* - create a procedure object
- *xvw_create_port()* - create a port object
- *xvw_create_loop()* - create a loop object
- *xvw_create_workspace()* - create a workspace object
- *xvw_create_toolboxmenu()* - create a toolbox menu object
- *xvw_create_commandbar()* - create a toolbox menu object
- *xvw_create_toolboxlist()* - create a toolbox list object
- *xvw_create_finderlist()* - create a finder list object

B. Basic Visual Programming Capabilities

The visual objects needed to support the most basic visual programming capabilities include the glyph object, the connection object, and the workspace object. The node object provides extensibility and flexibility within the visual programming toolkit. The port object is used by the glyph object to represent input and output connections within a visual program. The command bar object provides quick access to the most frequently used workspace object capabilities.

B.1. The Glyph Object

Figure 1: The glyph consists of a pane access button, a run button, data input connection nodes, data output connection nodes, and control connection nodes.

B.1.1. xvw_create_glyph() — *create a glyph object*

Synopsis

```
xvobject xvw_create_glyph(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: the name with which to reference the object

Returns

The glyph object on success, NULL on failure

Description

The purpose of the Glyph GUI object is to used to represent an operator within VisiQuest. Typically these are data processing routines that have a set of inputs and outputs that can be used to connect to other glyph's in which to form a visual program.

B.1.2. Attributes of the Glyph Object

Summary of Glyph Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_GLYPH_CLOSEPANE_PIXMAP	Pixmap	xvlang/misc/glyph/on.xpm	any bitmap or pixmap
XVW_GLYPH_CONNECTION_PARENT	xvobject	NULL	any xvobject that is subclassed off the ManagerClass
XVW_GLYPH_CONTROL_PIXMAP	Pixmap	xvlang/misc/glyph/control.xpm	any bitmap or pixmap
XVW_GLYPH_DAV_PIXMAP	Pixmap	xvlang/misc/glyph/dav.xpm	any bitmap or pixmap
XVW_GLYPH_DESTROY_PIXMAP	Pixmap	xvlang/misc/glyph/on.xpm	any bitmap or pixmap
XVW_GLYPH_DISTRIBUTE_PIXMAP	Pixmap	xvlang/misc/glyph/on.xpm	any bitmap or pixmap
XVW_GLYPH_ERROR_PIXMAP	Pixmap	xvlang/misc/glyph/error.xpm	any bitmap or pixmap
XVW_GLYPH_EXPRESSION_ID	long	0	and legal long value
XVW_GLYPH_FORM	kform *	NULL	a kform structure with a single pane
XVW_GLYPH_FORMFILE	char *	NULL	a UIS file with a single pane
XVW_GLYPH_INFO_PIXMAP	Pixmap	xvlang/misc/glyph/info.xpm	any bitmap or pixmap
XVW_GLYPH_OFF_PIXMAP	Pixmap	xvlang/misc/glyph/on.xpm	any bitmap or pixmap
XVW_GLYPH_ONAME	char *	NULL	any object within the toolbox specified by XVW_GLYPH_TBNAME
XVW_GLYPH_ON_PIXMAP	Pixmap	xvlang/misc/glyph/on.xpm	any bitmap or pixmap
XVW_GLYPH_OPENPANE_PIXMAP	Pixmap	xvlang/misc/glyph/on.xpm	any bitmap or pixmap
XVW_GLYPH_SHOWSTATUS	int	FALSE	TRUE/FALSE
XVW_GLYPH_TBNAME	char *	NULL	any legal toolbox
XVW_GLYPH_WKSPGUI	kform *	NULL	a kform structure with a single pane

Descriptions of Glyph Attributes	
Attribute	**Description**
XVW_GLYPH_CLOSEPANE_PIXMAP	The pixmap that is used to close the glyph's menuform
XVW_GLYPH_CONNECTION_PARENT	The parent in which connections between glyphs should be created. If NULL, then use the glyph's parent object.
XVW_GLYPH_CONTROL_PIXMAP	The pixmap that is used to indicate where control connections can be made.
XVW_GLYPH_DAV_PIXMAP	The pixmap that is used to indicate data available
XVW_GLYPH_DESTROY_PIXMAP	The pixmap that is used to destroy the glyph
XVW_GLYPH_DISTRIBUTE_PIXMAP	The pixmap that is used to distribute the location in which a glyph will be executed
XVW_GLYPH_ERROR_PIXMAP	The pixmap that is used to indicate that an error message needs to be displayed.

Descriptions of Glyph Attributes	
Attribute	**Description**
XVW_GLYPH_EXPRESSION_ID	The id in which expressions should be evaluated. Initially it uses id 0 so that all glyphs share, but if a glyph is placed within a workspace, then the workspace will have the id default to the workspace id itself.
XVW_GLYPH_FORM	The UIS form which is used to represent the glyph. If left NULL, then the XVW_GLYPH_TBNAME and XVW_GLYPH_ONAME will be used to retrieve the object's pane file.
XVW_GLYPH_FORMFILE	The UIS form file which is used to represent the glyph. If left NULL, then the XVW_GLYPH_TBNAME and XVW_GLYPH_ONAME will be used to retrieve the object's pane file.
XVW_GLYPH_INFO_PIXMAP	The pixmap that is used to indicate that an information message needs to be displayed.
XVW_GLYPH_OFF_PIXMAP	The pixmap that indicates the glyph is not running
XVW_GLYPH_ONAME	The object name in which the glyph is representing.
XVW_GLYPH_ON_PIXMAP	The pixmap that indicates the glyph is running
XVW_GLYPH_OPENPANE_PIXMAP	The pixmap that is used to open the glyph's menuform
XVW_GLYPH_SHOWSTATUS	Whether the status label beneath the glyph should appear.
XVW_GLYPH_TBNAME	The toolbox in which the object name can be found.
XVW_GLYPH_WKSPGUI	The form in which the glyph's user interface selections should be exported. If NULL then exporting selections is disabled.

B.1.3. Complete Resource Set of the Glyph Object

The complete resource set for the glyph object includes:

1. The glyph object attribute resource set, given in the previous section.

2. The node object attribute resource set, given in section G.2.

3. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3, "The *kwidgets* Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object."

4. The general object attributes, given in Chapter 2, "The *xvwidgets* Library," Section B, "General Attributes of GUI and Visual Objects."

B.2. The Workspace Object

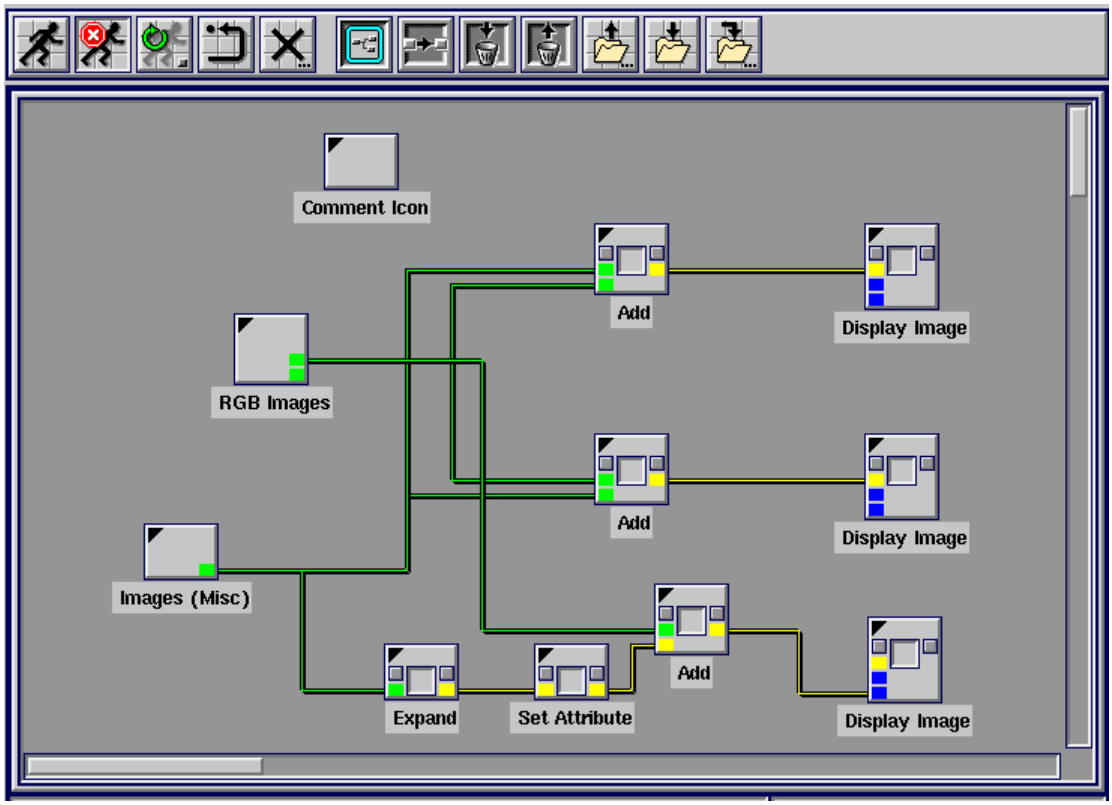


Figure 2: In this image of cantata, the command bar object appears above the workspace object. Here, a visual network has been created within the workspace.

B.2.1. xvw_create_workspace() — *create a workspace object*

Synopsis

```
xvobject xvw_create_workspace(
    xvobject parent,
    char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: the name with which to reference the object

Returns

The workspace object on success, NULL on failure

Description

The workspace visual programming object provides a canvas in which other visual programming objects may be placed and combined into a visual program. The workspace object supports the saving, restoring, execution, and management of visual programs. It also serves as a visual programming editor.

The workspace object does not directly support visual programming, but rather provides a canvas in which other visual programming components can be combined in which to create a visual program. Its role is similar to that of the manager object; however, instead of allowing the creation and management of GUI objects, the workspace object allows the creation and management of visual programming objects.

The workspace object allows (1) specification of an internal routine to be used for saving a network, and (2) specification of an internal routine to be used for restoring a network.

The workspace object can be used to create new visual programming models other than the one used in cantata; all that is necessary is to provide different routines for saving and restoring of networks.

Moreover, the workspace object has the capability to manipulate any other visual programming object that is subclassed from the node object, such as glyphs, conditionals, loops, and procedures. However, if the visual programming toolkit is extended to include other visual programming objects that are subclassed from the node object, they will automatically be usable in a workspace object.

As a visual programming editor, the workspace object comes with a full suite of capabilities including running a network, stopping a network, single stepping through a network, resetting the network, redrawing the network, clearing the network, checking the network, and providing information about the network.

B.2.2. Attributes of the Workspace Object

Summary of Workspace Attributes

Attribute (Resource Name)	Type	Default	Legal Values
XVW_WORKSPACE_ACTIVE_CALLBACK	kfunc_void	NULL	N/A
XVW_WORKSPACE_CHECK	int	N/A (read-only)	TRUE
XVW_WORKSPACE_CLEAR	int	N/A (read-only)	TRUE
XVW_WORKSPACE_CREATE_COUNTLOOP	int	N/A (read-only)	TRUE
XVW_WORKSPACE_CREATE_PROCEDURE	int	N/A (read-only)	TRUE
XVW_WORKSPACE_CREATE_WHILELOOP	int	N/A (read-only)	TRUE
XVW_WORKSPACE_ECHO	int	TRUE	TRUE/FALSE
XVW_WORKSPACE_EXPORT_SELECTIONS	int	N/A (read-only)	TRUE
XVW_WORKSPACE_FILENAME	char *	NULL	N/A

Summary of Workspace Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_WORKSPACE_INFO	int	N/A	TRUE
XVW_WORKSPACE_INHERIT	int	TRUE	TRUE/FALSE
XVW_WORKSPACE_NAME	char *	NULL	N/A
XVW_WORKSPACE_PARALLEL	int	TRUE	TRUE/FALSE
XVW_WORKSPACE_PROCEDURE_OBJECT	xvobject	created by the procedure object	any xvobject that is subclassed off the ProcedureClass
XVW_WORKSPACE_REDRAW	int	N/A (read-only)	TRUE
XVW_WORKSPACE_RESET	int	N/A (read-only)	TRUE
XVW_WORKSPACE_RESPONSIVE	int	TRUE	TRUE/FALSE
XVW_WORKSPACE_RESTORE	char *	N/A	any legal workspace file
XVW_WORKSPACE_RUN	int	N/A (read-only)	TRUE
XVW_WORKSPACE_RUN_CALLBACK	kfunc_void	NULL	N/A
XVW_WORKSPACE_SAVE	char *	N/A	any legal filename
XVW_WORKSPACE_SAVEALL	char *	N/A	any legal filename
XVW_WORKSPACE_SAVE_WKSPGUI	char *	N/A	any legal path to a file in which to store the UIS
XVW_WORKSPACE_SHOW_CLIPBOARD	int	N/A (read-only)	TRUE/FALSE
XVW_WORKSPACE_SHOW_WKSPGUI	int	N/A (read-only)	TRUE
XVW_WORKSPACE_STEP	int	N/A (read-only)	TRUE
XVW_WORKSPACE_STOP	int	N/A (read-only)	TRUE
XVW_WORKSPACE_VARIABLES	int	KWORKSPACE_VARIABLES_GLOBAL	KWORKSPACE_VARIABLES_GLOBAL KWORKSPACE_VARIABLES_LOCAL
XVW_WORKSPACE_WKSPGUI	kform *	$DESIGN/objects/library/xvlang/uis/panel.subform	any legal single pane kform structure

Descriptions of Workspace Attributes	
Attribute	**Description**
XVW_WORKSPACE_ACTIVE_CALLBACK	If desired, a callback may be installed on the workspace object that will be fired each time there is change to the active workspace. This helps to define the workspace scope to show which workspace is currently active.
XVW_WORKSPACE_CHECK	After a network is constructed, this action attribute may be used to "check" that all glyphs in the workspace have their required inputs and outputs specified.
XVW_WORKSPACE_CLEAR	After a network is constructed, this action attribute may be used to "clear" all glyphs in the workspace so that the workspace is empty.
XVW_WORKSPACE_CREATE_COUNTLOOP	This action attribute will take the currently selected glyphs within a workspace and place them inside of a newly created count loop.

Descriptions of Workspace Attributes	
Attribute	**Description**
XVW_WORKSPACE_CREATE_PROCEDURE	This action attribute will take the currently selected glyphs within a workspace and place them inside of a newly created procedure.
XVW_WORKSPACE_CREATE_WHILELOOP	This action attribute will take the currently selected glyphs within a workspace and place them inside of a newly created while loop.
XVW_WORKSPACE_ECHO	When set to 'true', this attribute causes the processes to be executed to be echoed to the console log. This attribute may be set to true or false.
XVW_WORKSPACE_EXPORT_SELECTIONS	After a network is constructed, this action attribute may be used to export the selections from the selected glyphs into the workspace GUI.
XVW_WORKSPACE_FILENAME	The workspace filename that represents the current workspace. This is used by the commandbar object to know where to save the current workspace to.
XVW_WORKSPACE_INFO	After a network is constructed, this action attribute may be used to display "information" on all glyphs can be displayed.
XVW_WORKSPACE_INHERIT	Procedures that are created from the top level workspace can have their own local attributes, or can inherit their attributes from the parent workspace. This attribute is made inactive on the toplevel parent. This attribute may be set to true or false.
XVW_WORKSPACE_NAME	The workspace name represents the name that is reflected by the procedure object's XVW_NODE_NAME.
XVW_WORKSPACE_PARALLEL	When a workspace contains several independant networks, turning parallel execution to 'true' makes the workspace execute quicker, as each of the independant networks will execute simulataneously. This attribute may be set to true or false.
XVW_WORKSPACE_PROCEDURE_OBJECT	The procedure object represented by the Workspace. By default this is NULL, but is necessary for the Workspace and Procedure to establish a protocol.
XVW_WORKSPACE_REDRAW	This action attribute is used to force a redisplay of the
XVW_WORKSPACE_RESET	After a network is constructed, this action attribute may be used to "touch" all glyphs in the workspace so that RUN will re-execute ALL glyphs, whether or not they
XVW_WORKSPACE_RESPONSIVE	When a workspace is put into 'responsive' mode then the workspace schedules glyphs such that scheduling of a network is continually done. When a workspace is running then the scheduler will continously monitor the network and run glyphs as they become modified. If the workspace is not running then the scheduler is turned off. By turning XVW_WORKSPACE_RESPONSIVE to FALSE the scheduler is always turned off.
XVW_WORKSPACE_RESTORE	This action attribute may be used to "restore" the workspace from the specified filename.
XVW_WORKSPACE_RUN	After a network is constructed, this action attribute may be used to execute the entire network.
XVW_WORKSPACE_RUN_CALLBACK	If desired, a callback may be installed on the workspace object that will be fired each time a glyph is run or stopped within the workspace.

Descriptions of Workspace Attributes	
Attribute	**Description**
XVW_WORKSPACE_SAVE	After a network is constructed, this action attribute may be used to "save" the workspace to the specified filename. Only the selected glyphs will be saved. If no glyphs are selected then all glyphs are then saved.
XVW_WORKSPACE_SAVEALL	After a network is constructed, this action attribute may be used to "save" the workspace to the specified filename. This differs from the XVW_WORKSPACE_SAVE attribute since it saves all glyphs, irregardless if they are selected or not.
XVW_WORKSPACE_SAVE_WKSPGUI	This action attribute can be used to save the workspace GUI parameters to a file. The filename is specified as part of the action attribute, which will contain a valid UIS specifi- workspace object, from the current workspace.
XVW_WORKSPACE_SHOW_CLIPBOARD	The clipboard is used to hold cut or copied items for later pasting. This this action attribute may be used to map or unmap the clipboard workspace.
XVW_WORKSPACE_SHOW_WKSPGUI	This action attribute can be used to force the workspace to map it's workspace GUI pane. If the pane is not realized then the pane is created and then mapped. This attribute can be used by the programmer to display the workspace GUI even if the workspace itself is not mapped. This allows the user to interact with the visual program via the workspace GUI parameters even though the workspace is not visible.
XVW_WORKSPACE_STEP	After a network is constructed, this action attribute may be used to single step execute a network.
XVW_WORKSPACE_STOP	If a workspace is currently running, this action attribute may be used to stop the execution of the entire network.
XVW_WORKSPACE_VARIABLES	Variables and expressions can be local to a procedure or global to the entire workspace. Setting 'Use Global Variables' to 'false' will cause the variables to only be known within the local procedure or workspace; setting available to all workspaces. This attribute may be set to KWORKSPACE_VARIABLES_GLOBAL or KWORKSPACE_VARIABLES_LOCAL
XVW_WORKSPACE_WKSPGUI	This attribute is a read/only attribute that points to the GUI form structure which contains the workspace GUI parameters.

B.2.3. Complete Resource Set of the Workspace Object

The complete resource set for the workspace object includes:

1. The workspace object attribute resource set, given in the previous section.

2. The canvas object attribute resource set, given in Chapter 3, "The *kwidgets Library*," Section E.2, "Attributes of the Canvas Object."

3. The viewport object attribute resource set, given in Chapter 3, "The *kwidgets Library*," Section K.2, "Attributes of the Viewport Object."

4. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3, "The *kwidgets* Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object."

5. The general object attributes, given in Chapter 2, "The *xvwidgets* Library," Section B, "General Attributes of GUI and Visual Objects."

B.3. The Node Object

B.3.1. xvw_create_node() — *create an node object*

Synopsis

```
xvobject xvw_create_node(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
parent of the node object; NULL will cause a default toplevel to be created automatically

`name`
name with which to reference node object

Returns

The node widget on success, NULL on failure

Description

The node object provides the base class which is used to create iconic representations of textual language routines. The node object is the object from which the *glyph* objects are subclassed, and should be used as the base class for any new visual programming objects that might be created for textual

code representation.

The node object consists of a small window with an optional pixmap in the middle, and a name that appears below the window. The node object supports interactive placement; the user can place the node using a shadow outline, or the (x, y) position may be specified by the application.

A node object allows (1) specification of the corresponding routine to be executed, and (2) rules for determining when that routine may be executed.

When the routine to be executed is specified as an external process, the Node object does "large grain" execution, or distributed execution of operators representing individual programs, as the Glyph object does. On the other hand, when the routine to be executed is specified as an internally defined function, the node object does ""fine grain" execution, or the calling of a subroutine or function internal to the visual programming language itself. Thus, the Node object can support both large grain and fine grain visual programming.

The node object can be used to create new iconic representations; thus, it allows the *xvlang* library to be extended to support visual programming constructs other than the ones currently used in cantata.

When the Node object is used as the base class for iconic representation of modules and the Workspace object is utilized as the visual editor, any new visual object that is written to be subclassed off the Node will automatically be supported by the existing system.

In addition, any such new visual programming objects will be capable of coexisting with existing visual programming objects. They can be combined in a visual network as desired; allowed combinations are only restricted by the visual programming model, as appropriate, but not by the visual programming objects themselves.

B.3.2. Attributes of the Node Object

Summary of Node Attributes

Attribute (Resource Name)	Type	Default	Legal Values
XVW_NODE_EXECUTE	int	FALSE	TRUE/FALSE
XVW_NODE_LEAVE	int	TRUE	TRUE/FALSE
XVW_NODE_MODIFIED	int	TRUE	TRUE/FALSE
XVW_NODE_MODIFIED_CALLBACK	kfunc_void	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_NODE_NAME	char *	NULL	any terminated string

Summary of Node Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_NODE_PIXMAP	Pixmap	NULL	Valid Pixmap structure
XVW_NODE_PIXMAPFILE	char *	NULL	The full path to a valid xpm or xbm file, defining the desired pixmap. Note that the path may contain $TOOLBOX.
XVW_NODE_PLACEMENT	int	TRUE	TRUE/FALSE
XVW_NODE_REMOTE_ENABLE	int	FALSE	TRUE/FALSE
XVW_NODE_RESET	int	TRUE	TRUE
XVW_NODE_RUNNABLE	int	FALSE	TRUE/FALSE
XVW_NODE_RUN_CALLBACK	kfunc_void	NULL	callback function, in the form: void callback_function xvobject object, kaddr client_data, kaddr call_data)
XVW_NODE_SHOW_DAV	int	TRUE	TRUE/FALSE
XVW_NODE_SHOW_MODIFIED	int	TRUE	TRUE/FALSE
XVW_NODE_TEST	int	FALSE	TRUE/FALSE
XVW_NODE_TYPE	int	KNODE_TYPE_UNKNOWN	KNODE_TYPE_UNKNOWN KNODE_TYPE_SOURCE KNODE_TYPE_TRANSFER KNODE_TYPE_SINK

Descriptions of Node Attributes	
Attribute	**Description**
XVW_NODE_EXECUTE	This *action attribute* executes a non-running node (set to TRUE), or stops a currently executing node (set to FALSE).
XVW_NODE_LEAVE	Indicates whether or not the node should be unmapped when its menu is being displayed; when set to TRUE the node is left visible, when set to FALSE, the node is unmapped.
XVW_NODE_MODIFIED	Indicates whether the node is in an updated state, or has been modified (and needs to be run). If TRUE, this attribute indicates that the node needs to be executed. When TRUE, this attribute is also used to clear the node's data available indicators.
XVW_NODE_MODIFIED_CALLBACK	If desired, a callback may be installed on the node that will be fired when the node becomes modified. See XVW_NODE_MODIFIED for details on node modified status.
XVW_NODE_NAME	The name with which to identify the node. The name will be printed below the node.

Descriptions of Node Attributes	
Attribute	**Description**
XVW_NODE_PIXMAP	This is the pixmap that appears in the center of the node. Candidates for the value of this attribute may be created with the use of *XCreatePixmap()*; see *The Xlib Reference Manual* by O'Reilly and Associates. Note that this attribute | is mutually exclusive with XVW_NODE_PIXMAPFILE; specify one or the other, not both.
XVW_NODE_PIXMAPFILE	This is the file defining the pixmap that appears in the center of the node.
XVW_NODE_PLACEMENT	Indicates whether or not the node should be placed interactively (TRUE) or non-interactively (FALSE).
XVW_NODE_REMOTE_ENABLE	If TRUE, this attribute indicates that the node is capable of being executed on a remote architecure.
XVW_NODE_RESET	This *action attribute* resets a node to it's initial state. After a node has been run, it is sometimes necessary to reset the node in order to get it re-run.
XVW_NODE_RUNNABLE	This *action attribute* allows the user to inquire if a node is ready to run. This is used by the workspace step execute to see which glyph should be run.
XVW_NODE_RUN_CALLBACK	If desired, a callback may be installed on the node that will be fired when the node is executed (turned ON) or stopped (turned OFF).
XVW_NODE_SHOW_DAV	If TRUE, this attribute indicates that the node should display a visual indication of when data is available at a port.
XVW_NODE_SHOW_MODIFIED	If TRUE, this attribute specifies that the node should visually indicate when it has been modified. See XVW_NODE_MODIFIED for details on node modified status.
XVW_NODE_TEST	This *action attribute* tests a non-running node (set to TRUE), or stops testing a currently executing node (set to FALSE). It is used in order to simulate running a network without actually running the network itself.
XVW_NODE_TYPE	This attribute is used to represent the node type within a network topology. The types of nodes are: KNODE_TYPE_UNKNOWN KNODE_TYPE_SOURCE KNODE_TYPE_TRANSFER KNODE_TYPE_SINK The source type is used to represent a node which has no inputs, but has optional outputs. The transfer type is a node which has both specified inputs and outputs. And finally the sink type is used to represent a node which has inputs, but no outputs.

B.3.3. Complete Resource Set of the Node Object

The complete resource set for the node object includes:

1. The node object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3, "The *kwidgets* Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object."

3. The general object attributes, given in Chapter 2, "The *xvwidgets* Library," Section B, "General Attributes of GUI and Visual Objects."

B.4. The Port Object

B.4.1. xvw_create_port() — *create a port object*

Synopsis

```
xvobject xvw_create_port(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: the name with which to reference the object

Returns

The port object on success, NULL on failure

Description

The purpose of the Port object is to provide a general mechanism for representing a "port" within a visual program. Typically these are used by the glyph to represent input and output connections within the visual program. Normally the port is created by other visual programming components, such as the Node object, but is provided so that other styles of visual programs can be created.

The Port object is really available to reduce the complexity of the application programmer in dealing with the user interactions of connecting Glyphs together. It also manages the complexity of representing the connections and the UIS that is used to represent the Port. So if the user updates the filename represented by the Port, and the Port is an output, then the contains of the Ports connected to the output, are updated appropriately.

B.4.2. Attributes of the Port Object

Summary of Port Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_PORT_CONNECTION_PARENT	xvobject	NULL	any xvobject that is subclassed off the ManagerClass
XVW_PORT_DAV	int	FALSE	TRUE/FALSE
XVW_PORT_DAV_CALLBACK	kfunc_void	NULL	any legal callback routine
XVW_PORT_DAV_PIXMAP	Pixmap	xvlang/misc/glyph/dav.xpm	any bitmap or pixmap
XVW_PORT_FILENAME	char *	NULL	any legal path and filename
XVW_PORT_FILENAME_CALLBACK	kfunc_void	NULL	any legal callback routine
XVW_PORT_MODIFIED	int	FALSE	TRUE/FALSE
XVW_PORT_OPTIONAL_PIXMAP	Pixmap	xvlang/misc/glyph/io_optional.xpm	any bitmap or pixmap
XVW_PORT_REQUIRED_PIXMAP	Pixmap	xvlang/misc/glyph/io_required.xpm	any bitmap or pixmap
XVW_PORT_SELECTED	int	FALSE	TRUE/FALSE
XVW_PORT_SELECTED_CALLBACK	kfunc_void	NULL	any legal callback routine
XVW_PORT_SELECTED_PIXMAP	Pixmap	xvlang/misc/glyph/io_selected.xpm	any bitmap or pixmap
XVW_PORT_SELECTION	kformstruct *	NULL	a legal KUIS_INPUTFILE or KUIS_OUTPUTFILE selection
XVW_PORT_SHOWDAV	int	TRUE	TRUE/FALSE
XVW_PORT_TYPE	int	KPORT_TYPE_UNKNOWN	KPORT_TYPE_UNKNOWN KPORT_TYPE_INPUT KPORT_TYPE_OUTPUT KPORT_TYPE_TRANSIENT

Descriptions of Port Attributes	
Attribute	**Description**
XVW_PORT_CONNECTION_PARENT	The parent in which connections between ports should be created. If NULL, then use the port's parent object.
XVW_PORT_DAV	Whether the port currently has data available or not. Data available is used to indicate whether the port is referencing valid data or not. This attribute is used in conjunction with the GlyphClass to coordinate the scheduling and execution of glyphs.
XVW_PORT_DAV_CALLBACK	Callback routines which will be notified when a port's data available state changes.
XVW_PORT_DAV_PIXMAP	The pixmap that used to visually indicate that a port's data available is present.

Descriptions of Port Attributes	
Attribute	**Description**
XVW_PORT_FILENAME	Filename associated with the port. Depending on whether will be propagated to other input ports.
XVW_PORT_FILENAME_CALLBACK	Callback routines which will be notified when a port's filename changes.
XVW_PORT_MODIFIED	Whether the port is current modified or not.
XVW_PORT_OPTIONAL_PIXMAP	The pixmap that is used to visually indicate that a port is optionally not selected.
XVW_PORT_REQUIRED_PIXMAP	The pixmap that is used to visually indicate that a port is selected.
XVW_PORT_SELECTED	Whether the port is currently optionally selected or not.
XVW_PORT_SELECTED_CALLBACK	Callback routines which will be notified when a port changes whether it is selected or not.
XVW_PORT_SELECTED_PIXMAP	The pixmap that is used to visually indicate that a port is selected for connecting with other ports.
XVW_PORT_SELECTION	The UIS selection that is used to represent the port. The selection needs to be either of type KUIS_INPUTFILE or KUIS_OUTPUTFILE.
XVW_PORT_SHOWDAV	Whether the visual representation for data available should be displayed when data is available. The pixmap used for representing data availabl can be specified by the XVW_PORT_DAV_PIXMAP resource.
XVW_PORT_TYPE	The type of port. The port type is used to indicate how the information of data available should be propagated. If the port is of type KPORT_TYPE_OUTPUT, then when data available, XVW_PORT_DAV, is set then the value will propagated to the other connections.

B.4.3. Complete Resource Set of the Port Object

The complete resource set for the port object includes:

1. The port object attribute resource set, given in the previous section.

2. The pixmap object attribute resource set, given in Chapter 3, "The *kwidgets* Library, section S.2, "Attributes of the Pixmap Object."

3. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3, "The *kwidgets* Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object."

4. The general object attributes, given in Chapter 2, "The *xvwidgets* Library," Section B, "General Attributes of GUI and Visual Objects."

B.5. The Command Bar Object

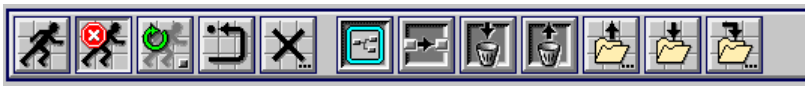


Figure 3: The commandbar object as used in cantata.

B.5.1. xvw_create_commandbar() — *create a toolbox menu object*

Synopsis

```
xvobject xvw_create_commandbar(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: the name with which to reference the object

Returns

The commandbar object on success, NULL on failure

Description

Creates a toolbox menu object, that consists of three menu objects. The first menu displays the different categories. The second menu displays the sub-categories for the selected category. And finally the third displays the operators for the selected sub-category.

Two callbacks are provided which can be used to notify the programmer when the user selects or activates an operator from the menu.

B.5.2. Attributes of the Command Bar Object

Summary of Command Bar Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_COMMANDBAR_WORKSPACE	kaddr	NULL	N/A

Descriptions of Command Bar Attributes	
Attribute	**Description**
XVW_COMMANDBAR_WORKSPACE	The workspace object which the commandbar object is associated with.

B.5.3. Complete Resource Set of the Command Bar Object

The complete resource set for the command bar object includes:

1. The command bar object attribute resource set, given in the previous section.

2. The rowcol object attribute resource set, given in Chapter 3, "The *kwidgets* Library," Section H.2, "Attributes of the RowCol Object."

3. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3, "The kwidgets Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object."

4. The general object attributes, given in Chapter 2, "The *xvwidgets* Library," Section B, "General Attributes of GUI and Visual Objects."

B.5.4. Example Using the Command Bar Object

```
#include <imagine.h>

static void quit_cb PROTO((xvobject, kaddr, kaddr));

/*
 *  This program demonstrates the use of the commandbar object, as well
 *  as the use of an announcement handler.
 *
 *  It creates a workspace with a commandbar and quit button,
 *  and restores the color arithmetic workspace "workspaces:ColorArith1".
 *  The commandbar object may be used to perform various operations,
 *  such as clearing the workspace, restoring another workspace, creating
 *  a loop or a procedure, deleting selected glyphs, and so on.
 *
```

```
 *  A labelstring object is created beneath the command bar which will
 *  be updated by the announcement handler which is installed using
 *  kset_announcehandler().  This allows the label string to
 *  echo what each pixmap button on the command bar does when the pointer
 *  is moved over the command bar.
 */
static int update_status PROTO((char *, char *, char *, char *, char *));
static void quit_cb      PROTO((xvobject, kaddr, kaddr));

xvobject status;
main(
   int  argc,
   char *argv[])
{
     xvobject parent, workspace, quit;
     char  *filename = "workspaces:ColorArith1";

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /*
      *  Create the manager.  Note that since the parent is NULL, a
      *  toplevel window will be created and the manager display placed
      *  inside.
      */
     parent = xvw_create_manager(NULL, "Parent");

     /*
      *  Create the quit button so that the user can quit
      */
        quit = xvw_create_button(parent, "Quit");
        xvw_set_attributes(quit,
                XVW_LABEL, "Quit",
                XVW_BELOW, NULL,
                XVW_LEFT_OF, NULL,
                XVW_CHAR_WIDTH, 6.0,
                XVW_CHAR_HEIGHT, 1.2,
                NULL);

      /*-- Create the status labelstring --*/
     status = xvw_create_label(parent, "Status");
        xvw_set_attributes(status,
                XVW_BORDER_WIDTH,    2,
                XVW_TACK_EDGE,       KMANAGER_TACK_HORIZ,
             XVW_LEFT_OF,         quit,
             XVW_CHAR_MIN_HEIGHT, 1.0,
                XVW_FORCE_REDISPLAY, TRUE,
                NULL);
     /*
      *  by using kset_announcehandler() to install the update_status()
      *  announcement handler, all calls to kannounce() made by the
      *  command bar object will be directed to the update_status() routine,
```

```
         *  which in turn will update the status labelstring.
         */
        kset_announcehandler(update_status);

        /*
         *  Create the workspace.  The XVW_WORKSPACE_RESTORE attribute is
         *  used to specify the workspace file to be restored.
         */
        workspace = xvw_create_workspace(parent, "workspace");
        xvw_set_attributes(workspace,
             XVW_BELOW,             status,
             XVW_WIDTH,             750,
             XVW_HEIGHT,            450,
             XVW_WORKSPACE_RESTORE, filename,
             NULL);
        xvw_add_callback(quit, XVW_BUTTON_SELECT, quit_cb, parent);

        /* display & run the program */
        xvf_run_form();
}

/* ARGSUSED */
static void quit_cb(
    xvobject object,
    kaddr client_data,
    kaddr call_data)
{
        xvobject parent = (xvobject) client_data;
        xvw_destroy(parent);
}

/* ARGSUSED */
static int update_status(
   char *toolbox,
   char *program,
   char *library,
   char *routine,
   char *message)
{
        xvw_set_attribute(status, XVW_LABEL, message);
        return(TRUE);
}
```

C. User Interface Components

Each VisiQuest 2001 program has an assigned *category*, *subcategory*, and *operator name*. The use of the category/subcategory/operator name convention imposes a hierarchy on the VisiQuest operators, and makes the process of finding a particular operator from the hundreds of available operators a much easier task. The toolbox menu, toolbox list, and finder list objects provide three different methods of glyph creation within the

visual programming language.

C.1. The ToolboxMenu Object

Figure 4: The ToolboxMenu displays a set of pulldown menus offering access to the different programs in the VisiQuest system, organized according to category and subcategory.

C.1.1. xvw_create_toolboxmenu() — *create a toolbox menu object*

Synopsis

```
xvobject xvw_create_toolboxmenu(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: the name with which to reference the object

Returns

The toolboxmenu object on success, NULL on failure

Description

The toolbox menu object allows the user to select an operator according to its category/subcategory/operator name. In cantata, the toolbox menu object is used as the first method for glyph creation; the Creates a toolbox menu object, that consists of three menu objects. The first menu displays the different categories. The second menu displays the sub-categories for the selected category. And finally the third displays the operators for the selected sub-category.

Two callbacks are provided which can be used to notify the programmer when the user selects or activates an operator from the menu.

C.1.2. Attributes of the ToolboxMenu Object

Summary of ToolboxMenu Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_TOOLBOXMENU_CREATE	kfunc_void	NULL	N/A

Descriptions of ToolboxMenu Attributes	
Attribute	**Description**
XVW_TOOLBOXMENU_CREATE	If desired, a callback may be installed on the toolboxmenu object that will be fired each time an entry is desired to be created by the user. This is done by the user selecting an entry from within the walking pulldown menus.

C.1.3. Complete Resource Set of the ToolboxMenu Object

The complete resource set for the toolboxmenu object includes:

1. The toolboxmenu object attribute resource set, given in the previous section.

2. The rowcol object attribute resource set, given in Chapter 3, "The *kwidgets* Library," Section H.2, "Attributes of the RowCol Object."

3. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3, "The *kwidgets* Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object."

4. The general object attributes, given in Chapter 2, "The *xvwidgets* Library," Section B, "General Attributes of GUI and Visual Objects."

C.1.4. Example Using the ToolboxMenu Object

```
#include <imagine.h>

/*
 *   This example demonstrates the use of the toolbox menu, a set of
 *   menu buttons and walking menus, which list the available operators
 *   in the VisiQuest system by category / subcategory / icon name.
 *
 *   An event handler is installed for creation of an operator
 *   that is selected from the toolboxmenu.
 *
```

```
 *   Use the menubuttons labeled with the category to display menus
 *   with subcategory listings.  Follow subcategory listings to display
 *   walking menus with the icon names of operators in that subcategory.
 *   Selecting an icon name from the walking menu will cause the cursor
 *   will change to the zen sign at which time you can move the
 *   pointer into the workspace and place a newly created glyph.
 */
static void toolboxmenu_create_cb PROTO((xvobject, kaddr, kaddr));

xvobject workspace;

int main(
   int  argc,
   char *argv[])
{
     xvobject parent, toolboxmenu;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror("example", "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /* create a toplevel to place toolboxmenu and workspace */
     parent = xvw_create_manager(NULL, "toplevel");

     /*
      * create the toolbox menu object, and install event
      * handlers to print information on selection of an operator, and
      * to create a glyph in the workspace.
      */
     toolboxmenu = xvw_create_toolboxmenu(parent, "toolboxmenu");
     xvw_set_attribute(toolboxmenu, XVW_TACK_EDGE, KMANAGER_TACK_HORIZ);
     xvw_add_callback(toolboxmenu, XVW_TOOLBOXMENU_CREATE,
                toolboxmenu_create_cb, NULL);

     workspace = xvw_create_workspace(parent, "workspace");
     xvw_set_attributes(workspace,
                  XVW_BELOW,  toolboxmenu,
                  XVW_WIDTH,  700,
                  XVW_HEIGHT, 600,
                  NULL);

     /* display & run the program */
     xvf_run_form();
}

void toolboxmenu_create_cb(
    xvobject object,
    kaddr client_data,
    kaddr call_data)

{
        xvobject  node;
        xvw_program_info *info = (xvw_program_info *) call_data;
```

```
        if (!info)
           return;

        node = xvl_create_node(workspace, info);
}
```

C.2. The ToolboxList Object

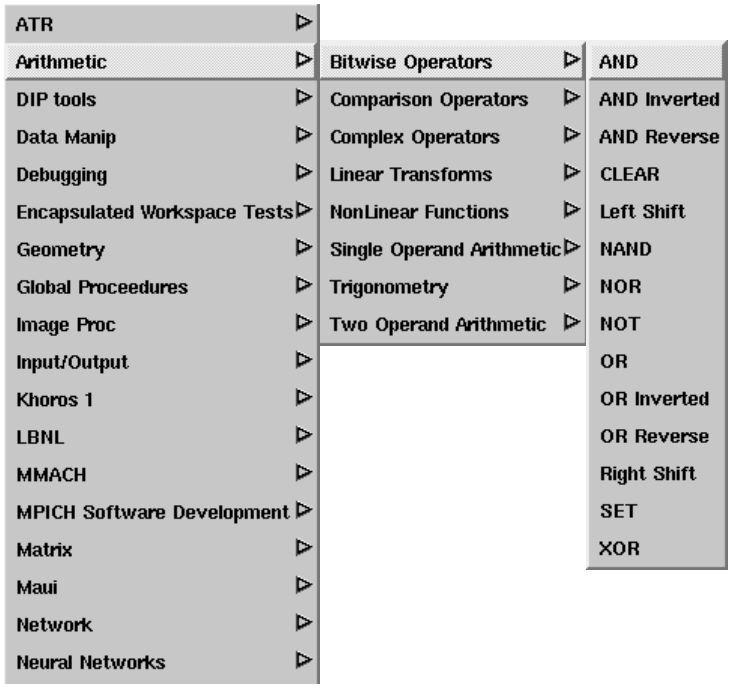


Figure 5: Here, the toolbox list shows the routines under the Arithmetic category and Bitwise Operators sub-category.

C.2.1. xvw_create_toolboxlist() — *create a toolbox list object*

Synopsis

```
xvobject xvw_create_toolboxlist(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: the name with which to reference the object

Returns

The toolboxlist object on success, NULL on failure

Description

Creates a toolbox list object, that consists of three list objects. The first list displays the different categories. The second list displays the sub-categories for the selected category. And finally the third displays the operators for the selected sub-category.

Two callbacks are provided which can be used to notify the programmer when the user selects or activates an operator from the list.

C.2.2. Attributes of the ToolboxList Object

Summary of FinderList Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_TOOLBOXLIST_CREATE	kfunc_void	NULL	N/A
XVW_TOOLBOXLIST_OPEN	int	N/A	TRUE
XVW_TOOLBOXLIST_SELECT	kfunc_void	NULL	N/A

Descriptions of ToolboxList Attributes	
Attribute	**Description**
XVW_TOOLBOXLIST_CREATE	If desired, a callback may be installed on the toolboxlist object that will be fired each time an entry is desired to be created by the user. This currently specified by either the user "double" clicking on an entry, or the programmer using the XVW_TOOLBOXLIST_OPEN *action attribute*.
XVW_TOOLBOXLIST_OPEN	This *action attribute* causes the XVW_TOOLBOXLIST_CREATE to be fired with the currently selected toolbox list entry.
XVW_TOOLBOXLIST_SELECT	If desired, a callback may be installed on the toolboxlist object that will be fired each time an entry is selected within the toolbox list.

C.2.3. Complete Resource Set of the ToolboxList Object

The complete resource set for the toolboxlist object includes:

1. The toolboxlist object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3, "The *kwidgets* Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object."

3. The general object attributes, given in Chapter 2, "The *xvwidgets* Library," Section B, "General Attributes of GUI and Visual Objects"

C.2.4. Example Using the ToolboxList Object

```
#include <imagine.h>

/*
 *   This example demonstrates the use of the toolbox list, a set of
 *   three list objects which list the available operators in the VisiQuest
 *   system by category / subcategory / icon name.
 *   Event handlers are installed for selection of an operator from the
 *   toolboxlist, and creation of an operator from the toolboxlist.
 *
 *   To use:
 *   1) Select a category, subcategory, or icon name from the toolboxlist.
 *      The information associated with the operator that corresponds to
 *      the currently selected category, subcategory, and icon name
 *      will be printed to the tty.
 *   2) Double click on an icon name in the toolboxlist.  The cursor
 *      will change to the zen sign at which time you can move the
 *      pointer into the workspace and place a newly created glyph.
 */
static void toolboxlist_select_cb PROTO((xvobject, kaddr, kaddr));
static void toolboxlist_create_cb PROTO((xvobject, kaddr, kaddr));

xvobject workspace;

main(
   int  argc,
   char *argv[])
{
     xvobject parent, toolboxlist;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "Unable to open display");
        kexit(KEXIT_FAILURE);
     }

     /*
      * create the toolbox list object, and install event
      * handlers to print information on selection of an operator, and
      * to create a glyph in the workspace.
      */
     toolboxlist = xvw_create_toolboxlist(NULL, "toolboxlist");
        xvw_set_attributes(toolboxlist,
          XVW_WIDTH,  300,
          XVW_HEIGHT, 200,
```

```
          NULL);
     xvw_add_callback(toolboxlist, XVW_TOOLBOXLIST_SELECT,
                toolboxlist_select_cb, NULL);
     xvw_add_callback(toolboxlist, XVW_TOOLBOXLIST_CREATE,
                toolboxlist_create_cb, NULL);

     workspace = xvw_create_workspace(NULL, "workspace");
     xvw_set_attributes(workspace,
                 XVW_WIDTH,  500,
                 XVW_HEIGHT, 600,
                 NULL);

     /* display & run the program */
     xvf_run_form();
}

void toolboxlist_select_cb(
    xvobject object,
    kaddr client_data,
    kaddr call_data)

{
     xvw_program_info *program_info = (xvw_program_info *) call_data;

     kfprintf(kstderr, "Program Info:\n");
     kfprintf(kstderr, "Toolbox name = %s\n",   program_info->tbname);
     kfprintf(kstderr, "Object  name = %s\n",   program_info->oname);
     kfprintf(kstderr, "Category     = %s\n",   program_info->category);
     kfprintf(kstderr, "Subcategory  = %s\n",   program_info->subcategory);
     kfprintf(kstderr, "Icon Name    = %s\n",   program_info->icon_name);
     kfprintf(kstderr, "Pane File    = %s\n",   program_info->panefile);
     kfprintf(kstderr, "Short Desc   = %s\n\n", program_info->short_desc);

}

void toolboxlist_create_cb(
    xvobject object,
    kaddr client_data,
    kaddr call_data)

{
     xvobject glyph;
        xvw_program_info *program_info = (xvw_program_info *) call_data;

     glyph = xvw_create_glyph(workspace, "glyph");
        xvw_set_attributes(glyph,
                XVW_NODE_NAME,      program_info->icon_name,
                XVW_GLYPH_FORMFILE, program_info->panefile,
                XVW_GLYPH_TBNAME,   program_info->tbname,
                XVW_GLYPH_ONAME,    program_info->oname,
                NULL);
}
```

C.3. The FinderList Object

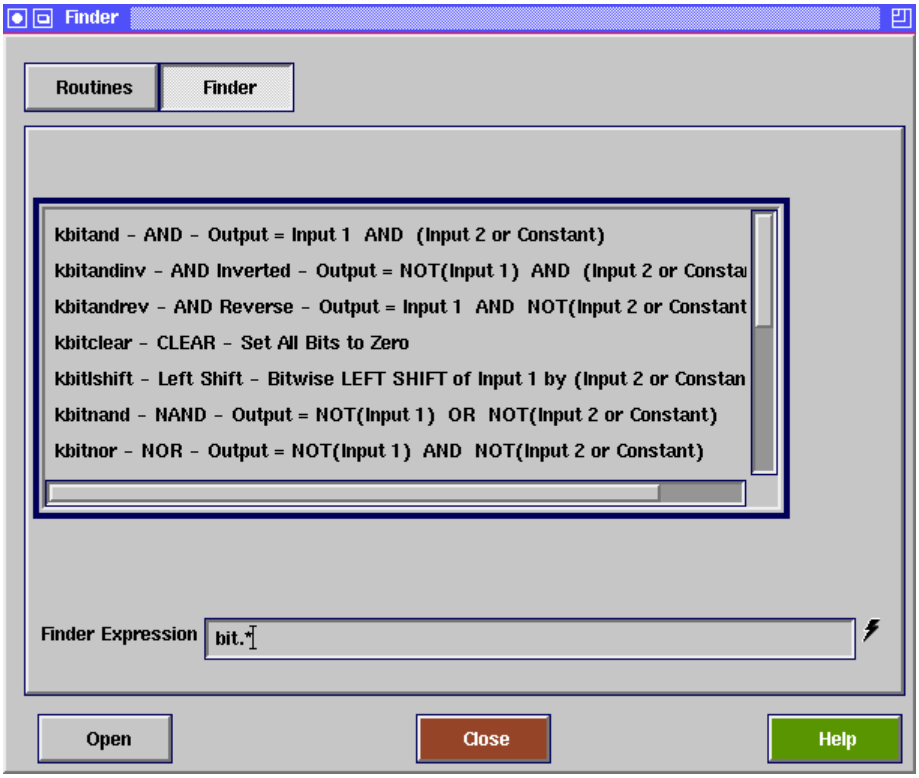


Figure 6: Here, the toolbox finder list shows the list of routines in the system with short descriptions that matches the expression "bit.*".

C.3.1. xvw_create_finderlist() — *create a finder list object*

Synopsis

```
xvobject xvw_create_finderlist(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: the name with which to reference the object

Returns

The finder list object on success, NULL on failure

Description

Creates a finder list object. The finder list is used to select an operator by name; key words can also be used to scan for a particular operator. The finder list consists of a single list containing all available operators; the user may scroll through the list and select the desired operator.

Two callbacks are provided which can be used to notify the programmer when the user selects or activates an operator from the list.

C.3.2. Attributes of the FinderList Object

Summary of FinderList Attributes

Attribute (Resource Name)	Type	Default	Legal Values
XVW_FINDERLIST_CREATE	kfunc_void	NULL	N/A
XVW_FINDERLIST_EXPRESSION	char *	NULL	any regular expression
XVW_FINDERLIST_OPEN	int	N/A	TRUE
XVW_FINDERLIST_SELECT	kfunc_void	NULL	N/A

Descriptions of FinderList Attributes

Attribute	Description
XVW_FINDERLIST_CREATE	If desired, a callback may be installed on the finderlist object that will be fired each time an entry is desired to be created by the user. This currently specified by either the user "double" clicking on an entry, or the programmer using the XVW_FINDERLIST_OPEN *action attribute*.

Descriptions of FinderList Attributes	
Attribute	**Description**
XVW_FINDERLIST_EXPRESSION	The regular expression used within the finder list. The regular expression which uses the following syntax. . Match any single character except newline * Match the preceding character or range of characters 0 or more times. The matching includes items within a [...]. [...] or [^..] Matches any one character contained within the brackets. If the first character after the '[' is the ']', then it is included in the characters to match. If the first character after the '[' is a '^', then it will match all characters NOT included in the []. The '-' will indicate a range of characters. For example, [a-z] specifies all characters between and including the ascii values 'a' and 'z'. If the '-' follows the '[' or is right before the ']' then it is interpreted literally. ^ If this is the first character of the regular expression, it matches the beginning of the line. $ If this is the last character of the regular expression, it matches the end of the line. \ This escapes the meaning of a special character.
XVW_FINDERLIST_OPEN	This *action attribute* causes the XVW_FINDERLIST_CREATE to be fired with the currently selected finder entry.
XVW_FINDERLIST_SELECT	If desired, a callback may be installed on the finderlist object that will be fired each time an entry is selected within the finderlist.

C.3.3. Complete Resource Set of the FinderList Object

The complete resource set for the finderlist object includes:

1. The finderlist object attribute resource set, given in the previous section.

2. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3, "The *kwidgets* Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object."

3. The general object attributes, given in Chapter 2, "The *xvwidgets* Library," Section B, "General Attributes of GUI and Visual Objects."

C.3.4. Example Using the FinderList Object

```
#include <imagine.h>

/*
 *   This example demonstrates the use of the finderlist, a list containing
 *   all available operators in the VisiQuest system in alphabetical order.
 *   First, a finderlist object is created to display the different
 *   operators available in the system; a textinput object underneath
 *   allows you to enter a key word with which to scan available operators.
 *   Event handlers are installed for selection of an operator from the
 *   finderlist, and creation of an operator from the finder list.
 *
 *   To use:
 *   1) Select an operator from the finder list.  It's associated
 *      information will be printed to the tty.
 *   2) Enter a scan string in the textinput selection below the finder
 *      list, such as "reverse" or "invert".  The finderlist will be
 *      updated with only those operators that include the specified
 *      string as part of their icon name, category, subcategory, or
 *      short description.
 *   3) Double click on an operator in the finder list.  The cursor
 *      will change to the zen sign at which time you can move the
 *      pointer into the workspace and place a newly created glyph.
 */
static void finderlist_select_cb (xvobject, kaddr, kaddr);
static void finderlist_create_cb (xvobject, kaddr, kaddr);
static void textinput_cb (xvobject, kaddr, kaddr);

xvobject workspace, parent;

void main(
   int  argc,
   char **argv)
{
     xvobject parent, text, finderlist;

        /* initialize VisiQuest program */
        khoros_initialize(argc, argv, "DESIGN");

     /* initialize the xvwidgets lib */
     if (!xvw_initialize(XVW_MENUS_XVFORMS))
     {
        kerror(NULL, "main", "unable to open display");
        kexit(KEXIT_FAILURE);
     }

     parent = xvw_create_manager(NULL, "finderlist");
     xvw_set_attributes(parent,
                XVW_MINIMUM_WIDTH,  400,
                XVW_MINIMUM_HEIGHT, 250,
```

```
                NULL);

      text = xvw_create_textinput(parent, "text");
         xvw_set_attributes(text,
           XVW_ABOVE, NULL,
           XVW_TACK_EDGE, KMANAGER_TACK_HORIZ,
           NULL);

      /*
       *  Create the workspace.  Note that since the parent is NULL, a
       *  toplevel window will be created and the workspace display placed
       *  inside.  The XVW_WORKSPACE_RESTORE attribute is used to specify
          *  the workspace file to be restored.
       */
      finderlist = xvw_create_finderlist(parent, "finderlist");
         xvw_set_attributes(finderlist,
           XVW_TACK_EDGE, KMANAGER_TACK_ALL,
           XVW_ABOVE, text,
                NULL);

      xvw_add_callback(finderlist, XVW_FINDERLIST_SELECT,
                finderlist_select_cb, NULL);
      xvw_add_callback(finderlist, XVW_FINDERLIST_CREATE,
                finderlist_create_cb, NULL);

      xvw_add_callback(text, XVW_TEXTINPUT_CALLBACK,
                textinput_cb, finderlist);

      workspace = xvw_create_workspace(NULL, "workspace");
      xvw_set_attributes(workspace,
                  XVW_WIDTH,  700,
                  XVW_HEIGHT, 800,
                  NULL);

      /* display & run the program */
      xvf_run_form();
}

void finderlist_select_cb(
    xvobject object,
    kaddr client_data,
    kaddr call_data)

{
      xvw_program_info *program_info = (xvw_program_info *) call_data;

      kfprintf(kstderr, "Program Info:\n");
      kfprintf(kstderr, "Toolbox name = %s\n",   program_info->tbname);
      kfprintf(kstderr, "Object  name = %s\n",   program_info->oname);
      kfprintf(kstderr, "Category     = %s\n",   program_info->category);
      kfprintf(kstderr, "Subcategory  = %s\n",   program_info->subcategory);
      kfprintf(kstderr, "Icon Name    = %s\n",   program_info->icon_name);
      kfprintf(kstderr, "Pane File    = %s\n",   program_info->panefile);
      kfprintf(kstderr, "Short Desc   = %s\n\n", program_info->short_desc);

}

void finderlist_create_cb(
    xvobject object,
```

```
    kaddr client_data,
    kaddr call_data)

{
     xvobject glyph;
        xvw_program_info *program_info = (xvw_program_info *) call_data;

     glyph = xvw_create_glyph(workspace, "glyph");
        xvw_set_attributes(glyph,
                XVW_NODE_NAME,      program_info->icon_name,
                XVW_GLYPH_FORMFILE, program_info->panefile,
                XVW_GLYPH_TBNAME,   program_info->tbname,
                XVW_GLYPH_ONAME,    program_info->oname,
                NULL);
}

void textinput_cb(
    xvobject object,
    kaddr client_data,
    kaddr call_data)

{
     xvobject finderlist = (xvobject) client_data;
     char *expr = *((char **) call_data);

     xvw_set_attribute(finderlist, XVW_FINDERLIST_EXPRESSION, expr);
}
```

D. Hierarchy

D.1. The Procedure Object

Figure 7: The procedure object.

D.1.1. xvw_create_procedure() — *create a procedure object*

Synopsis

```
xvobject xvw_create_procedure(
   xvobject parent,
```

```
    char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: the name with which to reference the object

Returns

The procedure object on success, NULL on failure

Description

The purpose of the procedure GUI object is to provide a visual heirarchy for visual programming. The procedure object consists of glyph object, which is the node used to represent the iconified visual program. When the Procedure is opened a workspace object will be mapped which will contain the visual program. The workspace can be used to construct visual programs, where the exported input/output parameters will appear on the Procedure object as input & output ports.

D.1.2. Attributes of the Procedure Object

Summary of Procedure Attributes

Attribute (Resource Name)	Type	Default	Legal Values
XVW_PROCEDURE_OPEN_PIXMAP	Pixmap	xvlang/misc/glyph/open.xpm	any bitmap or pixmap
XVW_PROCEDURE_PIXMAP	Pixmap	xvlang/misc/glyph/procedure.xpm	any bitmap or pixmap
XVW_PROCEDURE_WORKSPACE_OBJECT	xvobject	created by the procedure object	any xvobject that is subclassed off the ManagerClass

Descriptions of Procedure Attributes

Attribute	Description
XVW_PROCEDURE_OPEN_PIXMAP	The pixmap that is used to open a procedure glyph.
XVW_PROCEDURE_PIXMAP	The pixmap that is displayed in the center of the procedure.
XVW_PROCEDURE_WORKSPACE_OBJECT	The workspace object represented by the Procedure.

D.1.3. Complete Resource Set of the Procedure Object

The complete resource set for the procedure object includes:

1. The procedure object attribute resource set, given in the previous section.

2. The glyph object attribute resource set, given in section E.2.

3. The node object attribute resource set, given in section G.2.

4. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3, "The *kwidgets* Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object."

5. The general object attributes, given in Chapter 2, "The *xvwidgets* Library," Section B, "General Attributes of GUI and Visual Objects."

E. Control Flow

Control flow allows visual programs to branch and merge data flow, and to implement loops. The *xvlang* visual programming toolkit offers two types of objects that address the issues of control flow: the loop object and the conditional object. Both the Loop object and the Conditional object are special cases of (i.e., sub-classed from) the Glyph object.

E.1. The Conditional Object

Figure 8: Here, the conditional object is used to instantiate an if/else glyph.

E.1.1. xvw_create_conditional() — *create a conditional object*

Synopsis

```
xvobject xvw_create_conditional(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: the name with which to reference the object

Returns

The image object on success, NULL on failure

Description

The purpose of the Conditional GUI object is to provide a visual programming component in which to implement conditional flow constructs. Conditional flow constructs include such configurations as "if then else", "switch", etc.

E.1.2. Attributes of the Conditional Object

Summary of Conditional Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_CONDITIONAL_TYPE	int	N/A	N/A

Descriptions of Conditional Attributes	
Attribute	**Description**
XVW_CONDITIONAL_TYPE	Not available at this time.

E.1.3. Complete Resource Set of the Conditional Object

The complete resource set for the conditional object includes:

1. The conditional object attribute resource set, given in the previous section.
2. The glyph object attribute resource set, given in section E.2.
3. The node object attribute resource set, given in section G.2.
4. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3, "The *kwidgets* Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object."
5. The general object attributes, given in Chapter 2, "The *xvwidgets* Library," Section B, "General Attributes of GUI and Visual Objects."

E.2. The Loop Object

Figure 9: Here, the loop object is used to instantiate a while loop glyph.

E.2.1. xvw_create_loop() — *create a loop object*

Synopsis

```
xvobject xvw_create_loop(
   xvobject parent,
   char     *name)
```

Input Arguments

`parent`
: the parent object; NULL will cause a default toplevel to be created automatically

`name`
: the name with which to reference the object

Returns

The loop object on success, NULL on failure

Description

The purpose of the loop GUI object is to provide a visual heirarchy for visual programming. The loop object consists of glyph object, which is the node used to represent the iconified visual program. When the Loop is opened a workspace object will be mapped which will contain the visual program. The workspace can be used to construct visual programs, where the exported input/output parameters will appear on the Loop object as input & output ports.

E.2.2. Attributes of the Loop Object

Summary of Loop Attributes			
Attribute (Resource Name)	**Type**	**Default**	**Legal Values**
XVW_LOOP_TYPE	int	N/A	N/A

Descriptions of Loop Attributes	
Attribute	**Description**
XVW_LOOP_TYPE	Not available at this time.

E.2.3. Complete Resource Set of the Loop Object

The complete resource set for the loop object includes:

1. The loop object attribute resource set, given in the previous section.
2. The procedure object attribute resource set, given in section I.2.
3. The glyph object attribute resource set, given in section E.2.
4. The node object attribute resource set, given in section G.2.
5. The VisiQuest 2001 Manager attribute resource set, given in Chapter 3, The *kwidgets* Library," Section B.2, "Attributes of the VisiQuest 2001 Manager Object."
6. The general object attributes, given in Chapter 2, "The *xvwidgets* Library," Section B, "General Attributes of GUI and Visual Objects."

Table of Contents

Chapter 11

App-defaults

Chapter 11 - App-defaults

A. About app-defaults files

VisiQuest 2001 allows you to choose your own preferences. Suppose you prefer the GUI's of all applications to use a specific font. Perhaps you would like all 2D plots to come up with a default plot type of Discrete, or you like to see 2D axes displayed in green. You might decide that you like the pseudocolor object's palette to be of type colorwheel, or that you prefer the Spline connection type to be used to connect glyphs in VisiQuest.

All such preferences can be specified by setting *resources* in an application defaults file, or *app-defaults* file. There are two ways that resources can be set: (1) by the developer, in the code of the application, and (2) by the end user, in the appropriate *app-defaults* file.

App-defaults files are provided with VisiQuest 2001 to serve as models. You should not modify these app-defaults files. Instead, copy the files of interest to the directory `$HOME/.kri/KP2001/app-defaults` (you will have to create the app-defaults directory the first time you use it), and modify the copies as desired to specify your personal preferences.

There are two types of app-defaults files in VisiQuest 2001: *object* app-defaults files and *application* app-defaults files. Object app-defaults files are those that set the resources of entire classes of the different visual and GUI objects. For example, a resource setting for the button object in the Button app-defaults file will affect all buttons of all VisiQuest 2001 applications. The application app-defaults files are those that specify resources for a particular application.

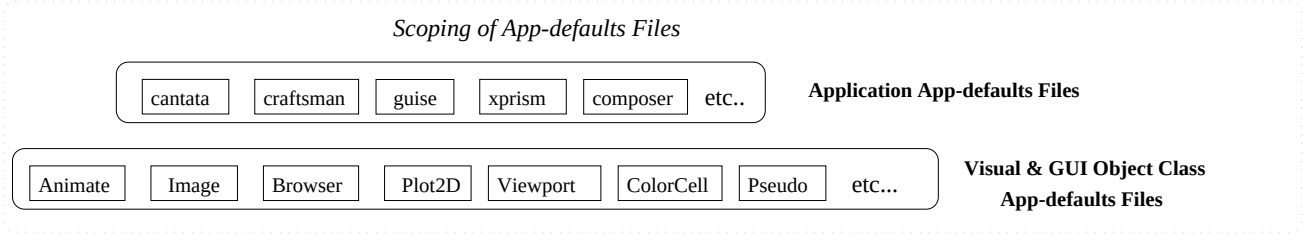


Figure 1: App-defaults files can specify resources by GUI or visual object, or by application.

A.1. GUI & Visual Object App-Defaults Files

There are a large number of object app-defaults files which specify resources by *class*. Every GUI and visual object available in the *kwidgets*, *xvobjects*, *xvisual*, and *xvlang* libraries of GUI & Visualization Services may have an associated app-defaults file. Template app-defaults files for GUI & visual objects may be found in one of the following directories,

```
$TOOLBOX/objects/library/kwidgets/app-defaults
$TOOLBOX/objects/library/xvobjects/app-defaults
$TOOLBOX/objects/library/xvisual/app-defaults
```

```
$TOOLBOX/objects/library/xvlang/app-defaults
```

depending on where the class definition for the object in question exists. App-defaults files for visual and GUI objects are named after the object. Filenames are capitalized, as in "Browser," "ColorCell," "Image," and "Plot2D." Note that the template object app-defaults files provided with VisiQuest 2001 are not actually used to set defaults; they are provided as models only.

GUI and visual object-specific, app-defaults files should include only those resource specifications which pertain to the class of the object in question.

A.2. Application Specific App-Defaults Files

Application app-defaults files specify resources for a particular application; there may be one app-defaults file for each xvroutine. Note that kroutines cannot have app-defaults files, since kroutines do not create GUI or visual objects.

The name of the application defaults file will be the same as the application name. The default app-defaults file for each application will be located in the app-defaults directory of the application. For example, the default app-defaults file for VisiQuest is be located in:

```
$TOOLBOX/objects/xvroutine/VisiQuest/app-defaults/VisiQuest
```

App-defaults files for applications are named after the application, as in guise, VisiQuest, and craftsman. Note that filenames are NOT capitalized. Note that the default application app-defaults files provided with the VisiQuest 2001 are really used to set defaults; modifications to these files will result in site-wide change in the appearance of the application.

Application-specific, app-defaults files should include only resource specifications for the corresponding application.

A.3. Precedence for Scoping of App-defaults Files

Application-specific, app-defaults files take precedence over object app-defaults files. User-specified, app-defaults files located in $HOME/.kri/KP2001/app-defaults take precedence over app-defaults files that are provided with VisiQuest 2001. For example, suppose you want *all* buttons to have a particular background color by default. It is easiest and most sensible to specify this in the "Button" app-defaults file, as opposed to setting the background color of buttons in the app-defaults file for every xvroutine. Suppose, however, that you only want the buttons in VisiQuest to have a particular color. In this case, you would specify the differing background color for buttons in the VisiQuest app-defaults file; the VisiQuest specification would over-ride the specification made in the "Button" app-defaults file, but all other xvroutines would continue to use the default.

Application-specific, app-defaults files specify the name of the application, whereas object app-defaults files specify only the name of the object class. For example, the entry:

```
*Button*background: grey
```

in the "Button" app-defaults file will make all buttons in every VisiQuest 2001 application have a background color of grey. Now, to over-ride the grey background setting for those buttons that appear in VisiQuest, you would add to the VisiQuest app-defaults file:

```
VisiQuest*Button*background: yellow
```

The result of having both app-defaults files will be that buttons in VisiQuest have a yellow background, while

all other buttons in VisiQuest 2001 have a grey background.

A.4. Creating Your Own App-Defaults files

It is *location* and *naming convention* that make app-defaults files work properly. When applications are initialized and objects are being created, GUI & Visualization Services will scan your home directory for:

```
$HOME/.kri/KP2001/app-defaults/
```

If the directory exists, *and* there is an app-defaults file inside named appropriately for the object being created or the application in question, the resources specified in that file will over-ride the defaults specified by VisiQuest 2001.

You may have as many or as few app-defaults files in your

```
$HOME/.kri/KP2001/app-defaults/
```

directory as you like.

For example, suppose you would like to specify resources for the applications guise and VisiQuest, and for the 2D plot object. You begin by creating an "app-defaults" directory in your home account. You then copy the four desired predefined app-defaults files to your directory. The following code segment shows the procedure to be followed:

```
% cd ~/.kri/KP2001
% mkdir app-defaults
% cd app-defaults
% cp $DESIGN/objects/xvroutine/VisiQuest/app-defaults/VisiQuest .
% cp $DESIGN/objects/xvroutine/guise/app-defaults/guise .
% cp $DESIGN/objects/library/xvisual/app-defaults/Plot2D .
```

Finally, you can modify your versions of the app-defaults files to specify resources as desired. The next time you execute guise or VisiQuest, or use an application that displays a 2D plot, the resources specified in your personal app-defaults files will take effect.

Important Note: If resource specifications in app-defaults files are made incorrectly, no error messages will be issued; the incorrect resource setting will simply have no effect.

A.5. Application Interaction with App-Defaults Files

As mentioned earlier, there are two ways that a resource may be specified. The developer of an application may specify the resource in the code of the application, or the end user of the application may specify the resource in an app-defaults file.

For example, suppose that two attributes of interest are the plot type in which a 2D plot will be displayed, and the foreground color in which it will appear. These resources might be specified by the developer as "Line Plot" and "yellow" in the application as follows:

```
xvw_set_attributes(plot2d_object,
                   XVW_PLOT2D_PLOTTYPE,    KPLOT2D_LINEPLOT,
                   XVW_FOREGROUND_COLOR, "yellow",
                   NULL);
```

Alternatively, the two resources might be specified by the end user as "Discrete" and "green" in the "Plot2D"

app-defaults file with:

```
*Plot2D.plot2DPlotType: 2
*Plot2D.foreground:     green
```

However, it is critical to understand that ONLY those resources NOT specified in the code of the application will take effect when set by the end user in the app-defaults file. Thus, in the example above, if the plot type and the foreground color were set *both* in the application and in the app-defaults file, the specification by the application would take precedence. You cannot over-ride resources that were set by the developer of an application. The plot would come up as a yellow line plot, regardless of the app-defaults file settings.

For this reason, it is recommended that all non-critical resource settings *not* be specified in the application. This allows the end user to specify resource settings as desired in an app-defaults file.

A.6. Issues With Regard to Specification of Object Resources

There are a few simple concepts used in object-oriented programming which should be understood in order to use app-defaults files efficiently and successfully. These include the concepts of inheritance, class-specific resources, and sub-parts of objects.

A.6.1. Inheritance & Class-Specific Resources

The GUI & visual objects offered by GUI & Visualization Services are designed in an object-oriented approach. This means that they *inherit* resources from the objects from which they are *subclassed*. In addition, they have their own *class-specific* resources. Class-specific resources are resources that are particular to the object in question; class-specific resources, in turn, will be inherited by any other objects that are subclassed from the object in question. Thus, app-defaults files for GUI & visual objects may contain settings not only for their own class-specific resources, but also for resources inherited from the objects from which they are subclassed.

In the following sections, the inheritance of each object is included in the app-defaults file, as in the following example using the PanIcon visual object:

```
INHERITANCE: manager -> graphics -> color -> image -> panicon
```

This may be read as, "The PanIcon object is subclassed from the Image object, which is subclassed from the Color class, which is subclassed from the Graphics class, which is subclassed from the Manager object." It implies that in the PanIcon app-defaults file, you may specify the following resources in order to control the operation and appearance of the panicon object:

- Class-specific resources of the PanIcon object
- Class-specific resources of the Image object
- Class-specific resources of the Color class

- Class-specific resources of the Graphics class
- Class-specific resources of the Manager object

In addition to the inheritance for the object, the app-defaults file also lists any class-specific resources for the object, as in the following example using the PanIcon:

```
CLASS-SPECIFIC RESOURCES: paniconSize
```

This states that the PanIcon object has only one class-specific resource of its own: the PanIcon size. However, resources of the Image object, Color class, Graphics class, and Manager object may also be specified in the PanIcon app-defaults file.

Thus, to get the full set of possible resources that can be set in an app-defaults file, you must read the inheritance listing of the object, and refer to the app-defaults resources listed for all the objects in the inheritance listing.

A.6.2. Precedence for Inheritance of Objects

Just as an object inherits attributes from the objects in its inheritance tree, it also inherits resource specifications from the app-defaults files of the objects in its inheritance tree. Thus, the PanIcon app-defaults file may not include a specification for the border width, but may inherit the specification for the border width specified for the Manager object in the Manager app-defaults file. You can, of course, over-ride inherited resource specifications simply by a new resource specification in the app-defaults files of the object in question. For example, if you want the PanIcon object to have a border width other than the default specified in the Manager app-defaults file, specifying the border width of the PanIcon directly will do the trick. Thus,

```
*PanIcon.borderWidth: 5
```

In the PanIcon app-defaults file will over-ride

```
*Manager.borderWidth: 2
```

in the Manager app-defaults file. When inherited resources are set in an app-defaults file, they will only affect the object in question, *not* the objects from which the object in question is sub-classed. Thus, if the border width resource inherited from the Manager object is specified in the PanIcon app-defaults file, the specification will take effect for panicon objects, but *not* for any other object.

A.6.3. Sub-Parts

Some GUI and visual objects contain other objects inside them; these subordinate objects are referred to as *sub-parts*. For example, the 2D Axis object contains two sub-parts: one axis object to represent the X axis, and another axis object to represent the Y axis.

When an object contains sub-parts, resources for those sub-parts may be specified in the app-defaults file. App-defaults settings for sub-parts must reference the *name* for the sub-part that was specified by the object when the sub-part was created. A listing of the sub-parts of an object, their names, and types are given in the app-defaults file when relevant, as follows:

```
!   SUB-PARTS: Sub-parts of the 2D Axis object include:
!
!   *Axis2D.xaxis    /* axis object,
!                       serves as the X axis */
!   *Axis2D.yaxis    /* axis object,
!                       serves as the Y axis */
!
!   You may specify resources for sub-parts of the axis object as desired.
!   See Axis app-defaults file for resources that can be set on
!   *Axis2D.xaxis and *Axis2D.yaxis.
```

For example, the foreground color of the *x* and *y* axis sub-parts of the 2D Axis object may be specified in the Axis2D app-defaults file as follows:

```
*Axis2D.xaxis.foreground: green
*Axis2D.yaxis.foreground: green
```

A.7. Syntax of the App-defaults File

Resources may be specified by the name of a specific instance of an object, by the class of the object, and by application. The dot, ".", is used to give the full name of a specific instance of an object, and to divide application names and class names from resource names. The wildcard symbol, "*", is used to simplify resource specifications. For a more detailed explanation of app-defaults file syntax, please see *The X Toolkit Intrinsics Programming Manual*, by Adrian Nye and Tim O'Reilly; rather than providing an exhaustive description of syntax specification, this section will use examples to illustrate the use of correct syntax in the app-defaults file. Note that when a resource specification has incorrect syntax, it will simply be ignored; the X Toolkit does not issue error messages when encountering incorrect specifications in app-defaults files.

To specify resources for an entire class of objects, the syntax:

**class_name.resource: value*

is used. For example, if the Marker app-defaults file had the specification:

```
*Marker.graphicsMarkertype: 10
```

this would cause *all* marker objects to use the hexagon marker type.

To specify resources for a particular application, simply add the application name at the front of the specification line, as in:

```
xprism*Marker.graphicsMarkertype: 8
```

This would over-ride the specification for the entire class of markers; thus, while all other markers would still use the hexagon marker type, markers displayed by xprism would use the diamond marker type instead.

Resource specifications may be made by object name, when the object name is known. In general, the only way to be sure of the name given to a specific instance of an object is to read the source code. By convention, however, all Help, License, and Quit buttons in VisiQuest 2001 are named "help," "license," and "quit." Therefore, you may specify the colors of these buttons for all applications using the wildcard specification followed by the name of the button in the appropriate widget-specific app-defaults file, as in:

```
*help.background:              forest green
*help.foreground:              white
```

```
*quit.background:               #ad5b5b
*quit.foreground:               white
*license.background:            goldenrod
*license.foreground:            white
```

Colors may be specified by name; valid color names may be found in /usr/lib/X11/rgb.txt. Alternatively, colors may be specified in their hexadecimal RGB numbers (for example, #ffff00 is full red, full green, no blue, and results in bright yellow).

When an object class has sub-parts (see Section A.4.3 above), the syntax:

**class_name.sub-part_name.resource: value*

must be used. For example, in order to make the palettes of any pseudocolor objects appear using the color-wheel palette type, the following specification would be made in the Pseudo app-defaults file:

```
*Pseudo.palette.paletteType: 3
```

Specifications of inherited resources for an object are made identically to specifications of class-specific resources for the object, and will over-ride specifications made for the class for which the resource was inherited. Suppose the String app-defaults file had the specification:

```
*String.stringEmphasize: false
```

This would cause string objects not to be emphasized. However, since the StringValue object is sub-classed from the String object, you may specify a differing value for the stringEmphasize resource in the StringValue app-defaults file, as in:

```
*StringValue.stringEmphasize: true
```

This would result in all stringvalue objects being emphasized, even though string objects would retain the specification of no emphasis as given in the String app-defaults file. Again, resources for a particular application may be specified differently if desired in the application specific app-defaults file, as in:

```
editimage*StringValue.stringEmphasize: false
```

This would cause stringvalue objects used by editimage not to be emphasized, although stringvalue objects in the rest of the system would still be emphasized because of the resource specification in the StringValue app-defaults file.

A.8. The Remainder of the Appendix

The remainder of this appendix gives excerpts from the predefined app-defaults files that are provided with VisiQuest 2001 for the GUI and visual objects in the *kwidgets*, *xvobjects*, *xvisual*, and *xvlang* app-defaults files. They are to be used as quick reference models for app-defaults files that you may create to define your own preferences. Filename, inheritance, class-specific resources, and sub-parts of each object are listed, followed by examples for settings of each class-specific resource for the object.

Comments in an app-defaults file are marked by an exclamation point ("!") in the first column of the file; you will notice that all lines in the predefined app-defaults files are commented out. The reason for this is that the predefined app-defaults files are not used to specify resources (the settings are all those that are the defaults anyway), but rather to be used as a reference. Remember that it is very expensive to set resources, and use of app-defaults files will result in increased startup time of an application.

For additional details on the app-defaults file, you may also see *The X Toolkit Intrinsics Programming Manual*, by Adrian Nye and Tim O'Reilly.

B. The Animate Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvimage/app-defaults/Animate`

Inheritance

manager -> graphics -> color -> image -> animate

Class-Specific Resources

animateUpdatetime
animateControl
animateDirection

Sub-Parts

```
The Animate object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding Attribute: XVW_ANIMATE_UPDATETIME
! Time interval, in seconds, which will elapse before the animation object
! is updated with the next image in the sequence.
!
!*Animate.animateUpdatetime: 1.0

!
! Corresponding attribute: XVW_ANIMATE_CONTROL
! See manual for explanation.  Values include:
!    KANIMATE_CONTROL_LOOP         1
!    KANIMATE_CONTROL_SINGLE       2
!    KANIMATE_CONTROL_AUTOREVERSE  3
!
!*Animate.animateControl: 1

!
! Corresponding attribute: XVW_ANIMATE_DIRECTION
! See manual for explanation. Values include:
!    KANIMATE_DIRECTION_STOP      1
!    KANIMATE_DIRECTION_PREVIOUS  2
!    KANIMATE_DIRECTION_NEXT      3
!    KANIMATE_DIRECTION_REVERSE   4
!    KANIMATE_DIRECTION_FORWARD   5
!
!*Animate.animateDirection: 1
```

C. The Area Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvplot/app-defaults/Area`

Inheritance

manager -> graphics -> area

Class-Specific Resources

areaDisplayTitle
areaDisplayDate

Sub-Parts

```
Sub-parts of the Area object include:
*Area.date     /* date object,
                  displays date in area object */
*Area.title    /* string object,
                  displays title in area object */
You may specify resources for sub-parts of the area object as desired.
See Date app-defaults file for resources that can be set on *Area.date,
See String app-defaults file for resources that can be set on *Area.title.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_AREA_DISPLAY_TITLE
! Whether or not the title is displayed in the area object.
!
!*Area.areaDisplayTitle: true

!
! Corresponding attribute: XVW_AREA_DISPLAY_DATE
! Whether or not the date is displayed in the area object.
!
!*Area.areaDisplayDate: true
```

D. The Axis Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvplot/app-defaults/Axis`

Inheritance

manager -> graphics -> axis

Class-Specific Resources

axisAxisMode
axisNumberMinorTics
axisTicJustification
axisShowAxis
axisShowAxisLabel
axisShowBox
axisShowMajorGrid
axisShowMinorGrid
axisShowNumericalLabel
axisShowTics
axisShowZeroLine
axisAxisColor
axisBoxColor
axisMajorGridColor
axisMinorGridColor
axisNumericalLabelsColor
axisMajorGridLineWidth
axisMinorGridLineWidth
axisMajorGridLineType
axisMinorGridLineType
axisNiceLabels

Sub-Parts

```
Sub-parts of the Axis object include:
*Axis.label    /* string object,
                  displays label of axis object */
You may specify resources for sub-parts of the axis object as desired.
See String app-defaults file for resources that can be set on *Axis.label.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_AXIS_AXIS_MODE
! Specifies use of linear or log scaling:
! KAXIS_LINEAR = 1, KAXIS_LOG = 2
!
!*Axis.axisAxisMode: 1

!
! Corresponding attribute: XVW_AXIS_NUMBER_MINOR_TICS
! Specifies the number of minor tics to appear on the axis
!
!*Axis.axisNumberMinorTics: 0

!
! Corresponding attribute: XVW_AXIS_TIC_JUSTIFICATION
! Determines how the major and minor axis tic marks appear:
! to the inside  of the axis line  (KAXIS_INSIDE   = 1)
! centered on the axis line        (KAXIS_CENTERED = 2),
! to the outside of the axis line  (KAXIS_OUTSIDE  = 3)
!
!*Axis.axisTicJustification: 1
```

```
!
! Corresponding attribute: XVW_AXIS_SHOW_AXIS
! Dictates whether or not the axis itself is displayed.
!
!*Axis.axisShowAxis: true

!
! Corresponding attribute: XVW_AXIS_SHOW_AXIS_LABEL
! Dictates whether or not the axis label is displayed.
!
!*Axis.axisShowAxisLabel: true

!
! Corresponding attribute: XVW_AXIS_SHOW_BOX
! Dictates whether or not the axis box will be displayed.
!
!*Axis.axisShowBox: true

!
! Corresponding attribute: XVW_AXIS_SHOW_MAJOR_GRID
! Dictates whether or not the major grid will be displayed.
!
!*Axis.axisShowMajorGrid: false


!
! Corresponding attribute: XVW_AXIS_SHOW_MAJOR_GRID
! Dictates whether or not the minor grid will be displayed.
!
!*Axis.axisShowMinorGrid: false

!
! Corresponding attribute: XVW_AXIS_SHOW_NUMERICAL_LABELS
! Dictates whether or not the numerical labels will be displayed.
!
!*Axis.axisShowNumericalLabel: true

!
! Corresponding attribute: XVW_AXIS_SHOW_TICS
! Dictates whether or not tic marks are displayed.
!
!*Axis.axisShowTics: true


!
! Corresponding attribute: XVW_AXIS_SHOW_ZERO_LINE
! Dictates whether or not zero line is displayed.
!
!*Axis.axisShowZeroLine: true


!
! Corresponding attribute: XVW_AXIS_AXIS_COLOR
! Name of color for axis
!
!*Axis.axisAxisColor: #00ff00

!
! Corresponding attribute: XVW_AXIS_BOX_COLOR
```

```
! Name of color for axis
!
!*Axis.axisBoxColor: #00ff00

!
! Corresponding attribute: XVW_AXIS_MAJOR_GRID_COLOR
! Name of color for major grid
!
!*Axis.axisMajorGridColor: #00ff00

!
! Corresponding attribute: XVW_AXIS_MINOR_GRID_COLOR
! Name of color for minor grid
!
!*Axis.axisMinorGridColor: #00ff00

!
! Corresponding attribute: XVW_AXIS_NUMERICAL_LABELS_COLOR
! Name of color for numerical labels
!
!*Axis.axisNumericalLabelsColor: #00ff00

!
! Corresponding attribute: XVW_AXIS_MAJOR_GRID_LINE_WIDTH
! The line width with which the major grid is drawn:
! KLINE_EXTRA_FINE     1
! KLINE_FINE           2
! KLINE_MEDIUM_FINE    3
! KLINE_MEDIUM         4
! KLINE_MEDIUM_WIDE    5
! KLINE_WIDE           6
! KLINE_EXTRA_WIDE     7
!
!*Axis.axisMajorGridLineWidth: 3

!
! Corresponding attribute: XVW_AXIS_MINOR_GRID_LINE_WIDTH
! The line width with which the minor grid is drawn, see values
! for line widths given for axisMajorGridLineWidth, above.
!
!*Axis.axisMinorGridLineWidth: 1


!
! Corresponding attribute: XVW_AXIS_MAJOR_GRID_LINE_TYPE
! The line type with which the major grid is drawn:
! KLINE_SOLID          1
! KLINE_DOTTED         2
! KLINE_DOT_DASH       3
! KLINE_SHORT_DASH     4
! KLINE_LONG_DASH      5
! KLINE_ODD_DASH       6
! KLINE_GRID_DOTTED    7
!
!*Axis.axisMajorGridLineType: 2

!
! Corresponding attribute: XVW_AXIS_MINOR_GRID_LINE_TYPE
! The line type with which the minor grid is drawn, see values
```

```
! for line types given for axisMajorGridLineType, above.
!
!*Axis.axisMinorGridLineType: 2

!
! Corresponding attribute: XVW_AXIS_NICE_LABELS
! See manual for description.
!
!*Axis.axisNiceLabels: false
```

E. The Axis2D Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvplot/app-defaults/Axis2D`

Inheritance

manager -> graphics -> axis2D

Class-Specific Resources

axis2dShowAxisX
axis2dShowAxisY

Sub-Parts

```
Sub-parts of the 2D Axis object include:
*Axis2D.xaxis    /* axis object,
                    serves as the X axis */
*Axis2D.yaxis    /* axis object,
                    serves as the Y axis */
You may specify resources for sub-parts of the axis object as desired.
See Axis app-defaults file for resources that can be set on
*Axis2D.xaxis and *Axis2D.yaxis.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_AXIS2D_SHOW_AXIS_X
!
!*Axis2D.axis2dShowAxisX: true

!
! Corresponding attribute: XVW_AXIS2D_SHOW_AXIS_Y
!
!*Axis2D.axis2dShowAxisY: true
```

F. The Browser Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/Browser`

Inheritance

manager -> browser

Class-Specific Resources

browserDirectory
browserFilter
browserDirectoryPixmapfile
browserAliasesPixmapfile
browserDestroyOnQuit

Sub-Parts

```
The Browser object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_BROWSER_DIRECTORY
! Directory from which the browser comes up, initially.
!
!*Browser.browserDirectory: "./"

! Corresponding attribute: XVW_BROWSER_FILTER
! Filter to be used with the list of files displayed by the browser.
!
!*Browser.browserFilter: "*.c"

! Corresponding attribute: XVW_BROWSER_DIRECTORY_PIXMAPFILE
! The pixmap that appears to the upper left of the browser object
! when the browser is in directory mode.
!
!*Browser.browserDirectoryPixmapfile: $DESIGN/objects/library/xvobjects/pixmaps/browse

! Corresponding attribute: XVW_BROWSER_ALIASES_PIXMAPFILE
! The pixmap that appears to the upper left of the browser object
! when the browser is in aliases mode.
!
!*Browser.browserAliasesPixmapfile: $DESIGN/objects/library/xvobjects/pixmaps/browser2

! Corresponding attribute: XVW_BROWSER_DESTROY_ON_QUIT
! Whether or not to destroy the browser object when the user selects
! a file or clicks on "Cancel".
!
!*Browser.browserDestroyOnQuit: true
```

G. The Canvas Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/Canvas`

Inheritance

manager -> viewport -> canvas

Class-Specific Resources

canvasGrid
canvasGridsize

Sub-Parts

```
The Canvas object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_CANVAS_GRID
! Specify when a grid should be displayed on the canvas, allowed values:
!
! KMANAGER_GRID_OFF    0: grid is always off.
! KMANAGER_GRID_ON     1: grid is always on.
! KMANAGER_GRID_EDIT   2: grid is only on when the canvas is in "edit mode"
! KMANAGER_GRID_SELECT 3: grid is only displayed when a child of the
|                         canvas has been selected by the user
!
!*Canvas.canvasGrid: 1

!
! Corresponding attribute: XVW_CANVAS_GRIDSIZE
! The size (in pixels) of the canvas grid
!
!*Canvas.canvasGridsize: 20

!
! Corresponding attribute: XVW_VP_ALLOW_HORIZ
!                         (Setting resource on *Canvas.viewport sub-part)
! Allow horizontal scrollbar?
!
!*Canvas.viewport.vpAllowHoriz: true

!
! Corresponding attribute: XVW_VP_ALLOW_VERT on viewport sub-part
!                         (Setting resource on *Canvas.viewport sub-part)
! Allow vertical scrollbar?
!
!*Canvas.viewport.vpAllowVert:   true

!
! Corresponding attribute: XVW_VP_FORCE_HORIZ on viewport sub-part
!                         (Setting resource on *Canvas.viewport sub-part)
```

```
! Insist on horizontal scrollbar (whether or not it is needed)?
!
!*Canvas.viewport.force_horizontal: true

!
! Corresponding attribute: XVW_VP_FORCE_VERT on viewport sub-part
!                          (Setting resource on *Canvas.viewport sub-part)
! Insist on vertical scrollbar (whether or not it is needed)?
!
!*Canvas.viewport.force_vertical:   true
```

H. The Circle Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvannotate/app-defaults/Circle`

Inheritance

manager -> graphics -> circle

Class-Specific Resources

none

Sub-Parts

```
The Circle object has no sub-parts.
```

I. The Color Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvisual/app-defaults/Color`

Inheritance

manager -> graphics -> color

Class-Specific Resources

colorNormType
colorNormUbyte
colorNormMethod
colorPrivateCmap

Sub-Parts

```
The Color class (a class, not an object) has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_COLOR_NORM_TYPE
! See manual for explanation.  Values include:
!    KCOLOR_NORM_GLOBAL 1
!    KCOLOR_NORM_LOCAL  2
!
!*colorNormType: 2

!
! Corresponding attribute: XVW_COLOR_NORM_UBYTE
! Specifies whether unsigned byte data should be normalized.
!
!*colorNormUbyte: true

!
! Corresponding attribute: XVW_COLOR_NORM_METHOD
! See manual for explanation. Values include:
!    KCOLOR_NORM_MAXCOLORS 1
!    KCOLOR_NORM_1STDDEV   2
!    KCOLOR_NORM_2STDDEV   3
!    KCOLOR_NORM_3STDDEV   4
!
!*colorNormMethod: 1

!
! Corresponding attribute: XVW_COLOR_PRIVATE_CMAP
! See manual for explanation.
!
!*colorPrivateCmap: false
```

J. The ColorCell Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvimage/app-defaults/ColorCell`

Inheritance

manager -> graphics -> color -> colorcell

Class-Specific Resources

colorcellShowindex
colorcellUpdateOnadd
colorcellRestoreOndelete

Sub-Parts

```
Sub-parts of the ColorCell object include:
*ColorCell.value    /* stringvalue object,
                       displyas the pixel value */
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_COLORCELL_SHOWINDEX
! Specifies whether pixel value associated with colorcell is to be displayed.
!
!*ColorCell.colorcellShowindex: true

!
! Corresponding attribute: XVW_COLORCELL_UPDATE_ONADD
! See manual for explanation.
!
!*ColorCell.colorcellUpdateOnadd: false

!
! Corresponding attribute: XVW_COLORCELL_RESTORE_ONDELETE
! See manual for explanation.
!
!*ColorCell.colorcellRestoreOndelete: false
```

K. The Connection Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/Connection`

Inheritance

manager -> connection

Class-Specific Resources

connectionType
connectionLinewidth
connectionUpdatetime

Sub-Parts

```
The Connection object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_CONNECTION_TYPE
! Specify connection type of connection object, choices include:
!
```

```
!  KCONNECTION_TYPE_LINEAR         1
!  KCONNECTION_TYPE_MANHATTAN      2
!  KCONNECTION_TYPE_SPLINE         3
!  KCONNECTION_TYPE_HEXAGON        4
!  KCONNECTION_TYPE_DIAMOND        5
!
!*Connection*connectionType:   2

!
! Corresponding attribute: XVW_CONNECTION_UPDATETIME
! Specify update time of connection object
!
*Connection*connectionUpdatetime:   0.05


!
! Corresponding attribute: XVW_CONNECTION_LINEWIDTH
! Specify line width of connection object, choices include:
!       KCONNECTION_LINE_EXTRA_FINE  0
!       KCONNECTION_LINE_FINE        1
!       KCONNECTION_LINE_MEDIUM_FINE 2
!       KCONNECTION_LINE_MEDIUM      3
!       KCONNECTION_LINE_MEDIUM_WIDE 4
!       KCONNECTION_LINE_WIDE        5
!       KCONNECTION_LINE_EXTRA_WIDE  6
!
!*Connection*connectionLinewidth:    2

!
! Corresponding attribute: XVW_FOREGROUND & XVW_BACKGROUND
!                          (Setting inherited resources)
! Specify foreground & background of connection object
!
!*Connection*foreground: #ffff00
!*Connection*background: #000000
```

L. The Console Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/Console`

Inheritance

manager -> viewport -> textdisplay -> console

Class-Specific Resources

none

Sub-Parts

The Console object has no sub-parts.

M. The Date Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvannotate/app-defaults/Date`

Inheritance

manager -> graphics -> string -> date

Class-Specific Resources

dateUpdatetime
dateFormat

Sub-Parts

```
The Date object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_DATE_UPDATETIME
! How often, in seconds, that the date is updated
!
!*Date.dateUpdatetime: 1

!
! Corresponding attribute: XVW_DATE_FORMAT
! The format in which to display the date
!
!*Date.dateFormat: "%h %d, 19%y %H:%M"
```

N. The Double Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/Double`

Inheritance

manager -> double

Class-Specific Resources

none

Sub-Parts

```
Sub-parts of the Double object include:
*Double.label     /* labelstring object,
```

```
                        displays label of double object                    */
*Double.text      /* text object,
                        provides parameter box in which user enters value */
*Double.cr_pixmap /* pixmap object,
                        indicates a "live" selection                       */
*Double.scrollbar /* scrollbar object,
                        which user can use to change value                 */
You may specify resources for sub-parts of the double object as desired.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_PIXMAP_FILENAME
!                          (setting resource on *Double.cr_pixmap sub-part)
! Specify pixmap to appear in upper left hand corner of double object
!
!*Double.cr_pixmap.pixmapFilename: pixmaps:lightning

!
! Corresponding attributes: XVW_FOREGROUND, XVW_BACKGROUND, XVW_FONTNAME
!                          (setting resources on *Double.label sub-part)
! Specify foreground color & font for label
!
!*Double.label.foreground: #000000
!*Double.label.labelEmphasize: 0
!*Double.label.fontName:       fixed

!
! Corresponding attributes: XVW_FOREGROUND, XVW_BACKGROUND, and XVW_FONTNAME
!                          (setting resources on *Double.text sub-part)
! Specify foreground, background color & font for text
!
!*Double.text.foreground: #000000
!*Double.text.background: #979797
!*Double.text.fontName:   fixed
!*Double.text.fontList:   fixed
```

O. The Error Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/Error`

Inheritance

manager -> error

Class-Specific Resources

errorPixmapfile

Sub-Parts

```
Sub-parts of the Error object include:
*Error.pixmap /* pixmap object,
                 decoration at upper left hand corner of Error object */
*Error.text   /* text object,
                 displays error message                               */
*Error.button /* button object,
                 acknowledgement button                               */
*Error.label  /* label object,
                 label at top of error object                         */
You may specify resources for sub-parts of the error object as desired.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_ERROR_PIXMAPFILE
! Specify pixmap to appear in upper left hand corner of error object
!
!*Error*errorPixmapfile: $DESIGN/objects/library/xvobjects/pixmaps/stopsign.xpm


!
! Corresponding attribute: XVW_FOREGROUND, XVW_FONTNAME
!                          (Setting resources on *Error.label sub-part)
! Specify foreground color & font for label of error object
!
!*Error.label.foreground: #000000
!*Error.label.fontName:       fixed
!*Error.label.fontList:   fixed

!
! Corresponding attribute: XVW_FOREGROUND, XVW_BACKGROUND, XVW_FONTNAME
!                          (Setting resources on *Error.button sub-part)
! Specify foreground, background color & font for button of error object
!
!*Error.button.foreground: #ffffff
!*Error.button.background: #ff0000
!*Error.button.fontName:       fixed
!*Error.button.fontList:   fixed


! Corresponding attribute: XVW_FOREGROUND, XVW_FONTNAME
!                          (Setting resources on *Error.text sub-part)
! Specify foreground & font for text of error object
!
!*Error.text.foreground: #000000
!*Error.text.fontName:       fixed
!*Error.text.fontList:   fixed
```

P. The Float Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/Float`

Inheritance

manager -> float

Class-Specific Resources

none

Sub-Parts

```
Sub-parts of the Float object include:
*Float.label     /* labelstring object,
                    label at left of float object                    */
*Float.text      /* text object,
                    provides parameter box in which user enters value */
*Float.cr_pixmap /* pixmap object,
                    indicating a "live" selection                    */
*Float.scrollbar /* scrollbar object,
                    which user can use to change value               */
You may specify resources for sub-parts of the float object as desired.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_PIXMAP_FILENAME
!                          (Setting resources on *Float.cr_pixmap sub-part)
! Specify pixmap to appear in upper left hand corner of float object
!
!*Float.cr_pixmap.pixmapFilename: pixmaps:lightning



!
! Corresponding attribute: XVW_FOREGROUND, XVW_LABEL_EMPHASIZE, XVW_FONTNAME
!                          (Setting resources on *Float.label sub-part)
! Specify foreground color & font for label of float object
!
!*Float.label.foreground: #000000
!*Float.label.labelEmphasize: 0
!*Float.label.fontName:           fixed

!
! Corresponding attribute: XVW_FOREGROUND, XVW_BACKGROUND, XVW_FONTNAME
!                          (Setting resources on *Float.text sub-part)
! Specify foreground, background color & font for button of float object
!
!*Float.text.foreground: #000000
!*Float.text.background: #979797
!*Float.text.fontName:       fixed
!*Float.text.fontList:   fixed
```

Q. The Help Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/Help`

Inheritance

manager -> help

Class-Specific Resources

helpDisplaytitle
helpDisplayquit
helpDisplaymenu
helpDestroyOnQuit
helpMoreFiles

Sub-Parts

```
Sub-parts of the Help object include:
*Help.quit        /* button object,
                     quit button on help object          */
*Help.title       /* label object,
                     displays label of help object       */
*Help.next        /* button object,
                     "next" button on help object        */
*Help.previous    /* button object,
                     "previous" button on help object    */
*Help.name        /* button object,
                     "name" button on help object        */
*Help.textdisplay /* textdisplay object,
                     used to display text                */
You may specify resources for sub-parts of the help object as desired.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_HELP_DISPLAYTITLE
! When set to false, this will suppress display of the title
!
!*Help.helpDisplaytitle: true

!
! Corresponding attribute: XVW_HELP_DISPLAYQUIT
! When set to false, this will suppress display of the quit button
!
!*Help.helpDisplayquit: true

!
! Corresponding attribute: XVW_HELP_DISPLAYMENU
! When set to false, this will suppress display of the options menu button
!
!*Help.helpDisplaymenu: true

!
```

```
! Corresponding attribute: XVW_HELP_DESTROY_ON_QUIT
! When set to false, this will cause the help object to be unmapped
! when the user clicks on the "Quit" button, rather than being destroyed
!
!*Help.helpDestroyOnQuit: true

!
! Corresponding attribute: XVW_HELP_MORE_FILES
! When set to false, help object will not stat the directory in order
! build the list of files included in the "Other Files" submenu
!
!*Help.helpMoreFiles: false
```

R. The Image Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvimage/app-defaults/Image`

Inheritance

manager -> graphics -> color -> image

Class-Specific Resources

imageXoffset
imageYoffset
imageBacking
imageRoiShape
imageRoiPolicy
imageRoiPresentation
imageRoiMultiband
imageRoiMultiple
imageComplexConvert

Sub-Parts

```
The Image object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_IMAGE_XOFFSET
! See manual for explanation.
!
!*Image.imageXoffset: -1

!
! Corresponding attribute: XVW_IMAGE_YOFFSET
! See manual for explanation.
!
!*Image.imageYoffset: -1
```

```
!
! Corresponding attribute: XVW_IMAGE_BACKING
! See manual for explanation.
!
!*Image.imageBacking: true

!
! Corresponding attribute: XVW_IMAGE_ROI_SHAPE
! See manual for explanation; values include:
!    KIMAGE_ROI_RECTANGLE  1
!    KIMAGE_ROI_POLYLINE   2
!    KIMAGE_ROI_CIRCLE     3
!    KIMAGE_ROI_ELLIPSE    4
!    KIMAGE_ROI_LINE       5
!    KIMAGE_ROI_FREEHAND   6
!    KIMAGE_ROI_CURVE      7
!
!*Image.imageRoiShape: 1

!
! Corresponding attribute: XVW_IMAGE_ROI_POLICY
! See manual for explanation; values include:
!       KIMAGE_ROI_INSIDE  1
!       KIMAGE_ROI_OUTLINE 2
!       KIMAGE_ROI_OUTSIDE 3
!
!*Image.imageRoiPolicy: 1

!
! Corresponding attribute: XVW_IMAGE_ROI_PRESENTATION
! See manual for explanation; values include:
!       KIMAGE_ROI_SIGNAL  1
!       KIMAGE_ROI_IMAGE   2
!       KIMAGE_ROI_SURFACE 3
!
!*Image.imageRoiPresentation: 2

!
! Corresponding attribute: XVW_IMAGE_ROI_MULTIBAND
! See manual for explanation.
!
!*Image.imageRoiMultiband: true

!
! Corresponding attribute: XVW_IMAGE_ROI_MULTIPLE
! See manual for explanation.
!
!*Image.imageRoiMultiple: false

!
! Corresponding attribute: XVW_IMAGE_COMPLEX_CONVERT
! See manual for explanation; values include:
!    KREAL       1
!    KIMAGINARY  2
!    KPHASE      3
!    KMAGNITUDE  4
!    KLOGMAGP1   5
!    KLOGMAG     6
!    KLOGMAGSQP1 7
```

```
!    KLOGMAGSQ   8
!    KMAGSQ      9
!
!*Image.imageComplexConvert: 5

!
! Corresponding attribute: XVW_MAXIMUM_WIDTH and XVW_MAXIMUM_HEIGHT
! This sets the maximum width and height of the image.
!
!*Image.maximumWidth:  1000
!*Image.maximumHeight: 1000
```

S. The ImageIcon Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvimage/app-defaults/ImageIcon`

Inheritance

manager -> graphics -> color -> image -> imageicon

Class-Specific Resources

imageiconSize

Sub-Parts

```
The ImageIcon object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_IMAGEICON_SIZE
! The size of the icon in pixels.
!
!*ImageIcon.imageiconSize: 100
```

T. The Indicator Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvplot/app-defaults/Indicator`

Inheritance

manager -> graphics -> marker -> indicator

Class-Specific Resources

indicatorConstraint
indicatorLine
indicatorShowXpos
indicatorShowYpos

Sub-Parts

```
The sub-parts of the Indicator object include:
*Indicator.xpos   /* stringvalue object,
                     displays x position of indicator                  */
*Indicator.ypos   /* stringvalue object,
                     displays y position of indicator                  */
*Indicator.xline  /* line object,
                     displays horizontal line for indicator            */
*Indicator.yline  /* line object,
                     displays vertical line for indicator              */
You may specify resources for sub-parts of the indicator object as desired.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_INDICATOR_CONSTRAINT
! See manual for explanation; values include:
!    KINDICATOR_CONSTRAINT_NONE 0
!       KINDICATOR_CONSTRAINT_X    1
!       KINDICATOR_CONSTRAINT_Y    2
!
!*Indicator.indicatorConstraint: 0

!
! Corresponding attribute: XVW_INDICATOR_LINE
! Indicates whether a line should be displayed; values include:
! #define KINDICATOR_LINE_NONE                0
! #define KINDICATOR_LINE_VERTICAL            1
! #define KINDICATOR_LINE_HORIZONTAL          2
! #define KINDICATOR_LINE_BOTH                3
!
!*Indicator.indicatorLine: 0

!
! Corresponding attribute: XVW_INDICATOR_SHOW_XPOS
! Iindicates whether the X position value should be displayed.
!
!*Indicator.indicatorShowXpos: true

!
! Corresponding attribute: XVW_INDICATOR_SHOW_YPOS
! Iindicates whether the Y position value should be displayed.
!
!*Indicator.indicatorShowYpos: true
```

U. The Info Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/Info`

Inheritance

manager -> info

Class-Specific Resources

infoPixmapfile

Sub-Parts

```
Sub-parts of the Info object include:
*Info.pixmap /* pixmap object,
                decorates upper left hand corner of Info object */
*Info.text   /* text object,
                displays info message                           */
*Info.button /* button object,
                acknowledgement button                          */
*Info.label  /* label object,
                label at top of info object                     */
You may specify resources for sub-parts of the info object as desired.
```

Resource Specifications in App-defaults File

```
! Corresponding Attribute: XVW_INFO_PIXMAPFILE
! Specify pixmap to appear in upper left hand corner of info object
!
!*Info*infoPixmapfile: $DESIGN/objects/library/xvobjects/pixmaps/info.xpm


!
! Corresponding Attribute: XVW_FOREGROUND, XVW_FONTNAME
!                          (Setting resources on *Info.label sub-part)
! Specify foreground color & font for label of info object
!
!*Info.label.foreground: #000000
!*Info.label.fontName:       fixed
!*Info.label.fontList:   fixed

!
! Corresponding Attribute: XVW_FOREGROUND, XVW_BACKGROUND, XVW_FONTNAME
!                          (Setting resources on *Info.button sub-part)
! Specify foreground, background color & font for button of info object
!
!*Info.button.foreground: #000000
!*Info.button.background: #ffff00
!*Info.button.fontName:       fixed
!*Info.button.fontList:   fixed


!
```

```
! Corresponding Attribute: XVW_FOREGROUND & XVW_FONTNAME
!                          (Setting resources on *Info.text sub-part)
! Specify foreground, background color & font for text of info object
!
!*Info.text.foreground: #000000
!*Info.text.fontName:       fixed
!*Info.text.fontList:   fixed
```

V. The InputFile Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/InputFile`

Inheritance

manager -> inputfile

Class-Specific Resources

none

Sub-Parts

```
Sub-parts of the InputFile object include:
*InputFile.cr_pixmap /* pixmap object,
                        represents a "live" selection               */
*InputFile.text      /* text object,
                        parameter box in which user enters value    */
*InputFile.button    /* button object,
                        displays label & brings up file browser     */
You may specify resources for sub-parts of the inputfile object as desired.
```

Resource Specifications in App-defaults File

```
! Corresponding Attribute: XVW_PIXMAP_FILENAME
!                          (Setting resources on *InputFile.cr_pixmap sub-part)
! Specify pixmap to appear in upper left hand corner of inputfile object
!
!*InputFile.cr_pixmap.pixmapFilename: pixmaps:lightning

!
! Corresponding Attribute: XVW_FOREGROUND, XVW_FONTNAME
!                          (Setting resources on *InputFile.button sub-part)
! Specify foreground & background color for button of inputfile object
!
!*InputFile.button.foreground: #000000
!*InputFile.button.fontName:       fixed
!*InputFile.button.fontList:   fixed

!
! Corresponding Attribute: XVW_FOREGROUND, XVW_BACKGROUND, XVW_FONTNAME
```

```
!                          (Setting resources on *InputFile.text sub-part)
! Specify foreground, background color & font for button of inputfile object
!
!*InputFile.text.foreground: #000000
!*InputFile.text.background: #979797
!*InputFile.text.fontName:       fixed
!*InputFile.text.fontList:   fixed
```

W. The InputOnly Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/InputOnly`

Inheritance

core -> inputonly

Class-Specific Resources

none

Sub-Parts

```
The InputOnly object has no sub-parts.
```

Resource Specifications in App-defaults File

```
!  Corresponding Attribute: XVW_CURSORNAME
!  The "Busy" inputonly object that is created specially by "xvw_busy()"
!  needs to have a special cursor to tell the user that the window is "busy".
!  Note that this cursor is NOT used by ALL InputOnly objects, only by the
!  Busy inputonly object.  If we wanted to specify the cursor for all
!  inputonly objects,  we would use "*InputOnly*cursor" instead.
!
*Busy*cursorName: $DESIGN/repos/bitmaps/cursors/zen
```

X. The Integer Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/Integer`

Inheritance

manager -> integer

Class-Specific Resources

none

Sub-Parts

```
Sub-parts of the Integer object include:
*Integer.label     /* labelstring object,
                      label on left of integer object          */
*Integer.text      /* text object,
                      parameter box in which user enters value  */
*Integer.cr_pixmap /* pixmap object,
                      indicates a "live" selection             */
*Integer.scrollbar /* scrollbar object,
                      which user can use to change value       */
You may specify resources for sub-parts of the integer object as desired.
```

Resource Specifications in App-defaults File

```
! Corresponding Attribute: XVW_PIXMAP_FILENAME
!                          (Setting resources on *Integer.cr_pixmap sub-part)
! Specify pixmap to appear in upper left hand corner of integer object
!
!*Integer.cr_pixmap.pixmapFilename: pixmaps:lightning


!
! Corresponding Attribute: XVW_FOREGROUND, XVW_LABEL_EMPHASIZE, XVW_FONTNAME
!                          (Setting resources on *Integer.label sub-part)
! Specify foreground color & font for label of integer object
!
!*Integer.label.foreground: #000000
!*Integer.label.labelEmphasize: 0
!*Integer.label.fontName:           fixed
!*Integer.label.fontList:       fixed

!
! Corresponding Attribute: XVW_FOREGROUND, XVW_BACKGROUND, XVW_FONTNAME
!                          (Setting resources on *Integer.text sub-part)
! Specify foreground, background color & font for button of integer object
!
!*Integer.text.foreground: #000000
!*Integer.text.background: #979797
!*Integer.text.fontName:       fixed
!*Integer.text.fontList:   fixed
```

Y. The Label Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/Label`

Inheritance

manager -> label

Class-Specific Resources

labelFilled
labelEmphasize
forceRedisplay

Sub-Parts

```
The LabelString object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding Attribute: XVW_LABEL_FILLED
! Set to true, this attribute causes the background of the
| labelstring to filled with the background color.
!
!*Label.labelFilled: false

!
! Corresponding Attribute: XVW_LABEL_EMPHASIZE
! Set to true, this attribute causes the label to be drawn twice,
! giving it a 3D, emphasized effect.
!
!*Label.labelEmphasize: false

!
! Corresponding Attribute: XVW_FORCE_REDISPLAY
! Set to true, this attribute  used by the labelstring object to achieve
! smoother, faster update of the label.
!
!*Label.forceRedisplay: false
```

Z. The Layout Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/Layout`

Inheritance

manager -> layout

Class-Specific Resources

layoutNumberAcross
layoutBufferSize
layoutBorderSize
layoutAreaJustification

Sub-Parts

The Layout object has no sub-parts.

Resource Specifications in App-defaults File

```
! Corresponding Attribute: XVW_LAYOUT_NUMBER_ACROSS
! Number of child objects to lay out in one row
!
!*Layout.layoutNumberAcross: 1

!
! Corresponding Attribute: XVW_LAYOUT_BUFFER_SIZE
! Distance between child objects in pixels
!
!*Layout.layoutBufferSize:5

!
! Corresponding Attribute: XVW_LAYOUT_BORDER_SIZE
! Line width of border of currently selected child
!
!*Layout.layoutBorderSize: 2

!
! Corresponding Attribute: XVW_LAYOUT_AREA_JUSTIFICATION
! How the last row of child objects is laid out, values:
! KLAYOUT_AREA_CENTER - 1
! KLAYOUT_AREA_RIGHT  - 2
! KLAYOUT_AREA_LEFT   - 3
! KLAYOUT_AREA_FULL   - 4
!
!*Layout.layoutAreaJustification: 1
```

AA. The Line Object

Generated From App-defaults file:

$ENVISION/objects/library/xvannotate/app-defaults/Line

Inheritance

manager -> graphics -> line

Class-Specific Resources

none

Sub-Parts

The Line object has no sub-parts.

AB. The Marker Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvannotate/app-defaults/Marker`

Inheritance

manager -> graphics -> marker

Class-Specific Resources

none

Sub-Parts

```
The Marker object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_GRAPHICS_MARKERTYPE
!                          (setting inherited resource)
! The marker type; values include:
!    KMARKER_NONE            0
!    KMARKER_ARC             1
!    KMARKER_BOW_TIE         2
!    KMARKER_BOX             3
!    KMARKER_CARET           4
!    KMARKER_CIRCLE          5
!    KMARKER_CROSS           6
!    KMARKER_DAGGER          7
!    KMARKER_DIAMOND         8
!    KMARKER_DOT             9
!    KMARKER_HEXAGON         10
!    KMARKER_POINT           11
!    KMARKER_SQUARE          12
!    KMARKER_TRIANGLE        13
!    KMARKER_V               14
!    KMARKER_X               15
!
!*Marker.graphicsMarkertype: 12
```

AC. The NotifyWindow Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/NotifyWindow`

Inheritance

manager -> notifywindow

Class-Specific Resources

notifywindowTitle
notifywindowLabel
notifywindowPixmapfile

Sub-Parts

```
Sub-parts of the NotifyWindow object include:
*NotifyWindow.pixmap /* pixmap object,
                        decorates upper left hand corner     */
*NotifyWindow.label  /* labelstring object,
                        label at top of notifywindow object  */
*NotifyWindow.text   /* labelstring object,
                        displays message                     */
You may specify resources for sub-parts of notifywindow object as desired.
```

Resource Specifications in App-defaults File

```
! Corresponding Attribute: XVW_NOTIFYWINDOW_PIXMAPFILE
! Specify pixmap to appear in upper left hand corner of notifywindow object
! *NotifyWindow*notifywindowPixmapfile: $DESIGN/objects/library/xvobjects/pixmaps/noti

!
! Corresponding Attribute: XVW_NOTIFYWINDOW_TITLE
! Specify title to appear in titlebar of notifywindow object
! (note: this is usually not set in the app-defaults file, but by the
!        application, with the name of the application using the notifywindow)
!
!*NotifyWindow*notifywindowTitle: "Notification"

!
! Corresponding Attribute: XVW_NOTIFYWINDOW_LABEL
! Specify label to appear at top of notifywindow object
!
!*NotifyWindow*notifywindowLabel: "Working..."


!
! Corresponding Attribute: XVW_FOREGROUND, XVW_BACKGROUND, XVW_FONTNAME,
!                          XVW_LABEL_EMPHASIZE
!                          (Setting resources on *NotifyWindow.label sub-part)
! Specify foreground color & font for label of notifywindow object
!
*NotifyWindow.label.labelEmphasize: false
*NotifyWindow.label.foreground: #000000
*NotifyWindow.label.fontName:       variable


!
! Corresponding Attribute: XVW_FOREGROUND, XVW_BACKGROUND, XVW_FONTNAME,
!                          XVW_LABEL_EMPHASIZE
!                          (Setting resources on *NotifyWindow.text sub-part)
! Specify foreground, background color & font for text of notifywindow object
!
```

```
*NotifyWindow.text.foreground: #000000
*NotifyWindow.text.fontName:       variable
*NotifyWindow.text.labelEmphasize: false
```

AD. The OutputFile Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/OutputFile`

Inheritance

manager -> outputfile

Class-Specific Resources

none

Sub-Parts

```
Sub-parts of the OutputFile object include:
*OutputFile.cr_pixmap /* pixmap object,
                         indicates a "live" selection                 */
*OutputFile.text      /* text object,
                         text parameter box in which user enters value */
*OutputFile.button    /* button object,
                         displays label & brings up file browser      */
You may specify resources for sub-parts of the outputfile object as desired.
```

Resource Specifications in App-defaults File

```
! Corresponding Attribute: XVW_PIXMAP_FILENAME
!                          (Setting resource on *OutputFile.cr_pixmap sub-part)
! Specify pixmap to appear in upper left hand corner of outputfile object
!
!*OutputFile.cr_pixmap.pixmapFilename: pixmaps:lightning

!
! Corresponding Attribute: XVW_FOREGROUND, BACKGROUND, XVW_FONTNAME
!                          (Setting resource on *OutputFile.button sub-part)
! Specify foreground color & font for button of outputfile object
!
!*OutputFile.button.foreground: #000000
!*OutputFile.button.background: #ffffff
!*Outputfile.button.fontName:       fixed
!*OutputFile.button.fontList:   fixed

!
! Corresponding Attribute: XVW_FOREGROUND, XVW_BACKGROUND, XVW_FONTNAME
!                          (Setting resource on *OutputFile.text sub-part)
! Specify foreground, background color & font for button of outputfile object
!
```

```
!*OutputFile.text.foreground: #000000
!*OutputFile.text.background: #979797
!*OutputFile.text.fontName:       fixed
!*OutputFile.text.fontList:   fixed
```

AE. The Palette Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvimage/app-defaults/Palette`

Inheritance

manager -> graphics -> color -> palette

Class-Specific Resources

paletteType

Sub-Parts

```
The Palette object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_PALETTE_TYPE
! The palette type, one of:
!     KPALETTE_TYPE_COLORBAR   1
!        KPALETTE_TYPE_COLORCELL  2
!        KPALETTE_TYPE_COLORWHEEL 3
!
!*Palette.paletteType: 1
```

AF. The PanIcon Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvimage/app-defaults/PanIcon`

Inheritance

manager -> graphics -> color -> image -> panicon

Class-Specific Resources

paniconSize

Sub-Parts

```
The PanIcon object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_PANICON_SIZE
! Desired physical size of pan icon, in pixels. Note that the actual size of
! the pan icon may be automatically modified to preserve proportionality in
! the event that a non-square image is displayed.
!
!*PanIcon.paniconSize: 100
```

AG. The Pixmap Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/Pixmap`

Inheritance

manager -> pixmap

Class-Specific Resources

pixmapFilename
pixmapMaskname

Sub-Parts

```
The Pixmap object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding Attribute: XVW_PIXMAP_FILENAME
! Name of file defining the desired pixmap
!
!*Pixmap*pixmapFilename: $TOOLBOX/...../filename.xpm

!
! Corresponding Attribute: XVW_PIXMAP_MASKNAME
! Name of file defining the desired mask for the pixmap
!
!*Pixmap*pixmapMaskname: $TOOLBOX/......./filename.xpm
```

AH. The Plot2D Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvplot/app-defaults/Plot2D`

Inheritance

manager -> graphics -> color -> plot2D

Class-Specific Resources

plot2DPlotType

Sub-Parts

```
The Plot2D object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_PLOT2D_PLOTTYPE
! Plot type; one of:
!       KPLOT2D_LINEPLOT           1
!       KPLOT2D_DISCRETE           2
!       KPLOT2D_BARGRAPH           3
!       KPLOT2D_POLYMARKER         4
!       KPLOT2D_LINEMARKER         5
!       KPLOT2D_COLORMARKER        6
!
!*Plot2D.plot2DPlotType: 1
```

AI. The Plot3D Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvplot/app-defaults/Plot3D`

Inheritance

manager -> graphics -> color -> plot3D

Class-Specific Resources

plot3DPlotType
plot3DShadeType

Sub-Parts

```
The Plot3D object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_PLOT3D_PLOTTYPE
! Plot type; one of:
!       KPLOT3D_LINEPLOT           20
!       KPLOT3D_WIREFRAME          21
!       KPLOT3D_MESH               22
!       KPLOT3D_COLORMESH          23
!       KPLOT3D_HORIZON            24
!       KPLOT3D_SCATTER            25
!       KPLOT3D_IMPULSE            26
!       KPLOT3D_CONTOUR_2D         27
!       KPLOT3D_CONTOUR_3D         28
!       KPLOT3D_CONSTANT_SHADING   29
!       KPLOT3D_PHONG_SHADING      30
!       KPLOT3D_GHOURAUD_SHADING   31
!       KPLOT3D_3D_BEZIER          32
!       KPLOT3D_SURFACE_BEZIER     33
!       KPLOT3D_RENDERED_BEZIER    34
!
!*Plot3D.plot3DPlotType: 20
```

```
! Corresponding attribute: XVW_PLOT3D_SHADETYPE
! Plot type; one of:
!       KPLOT3D_SHADE_IMAGERY       1
!       KPLOT3D_SHADE_ELEVATION     2
!       KPLOT3D_SHADE_NORMAL        3
!
!*Plot3D.plot3DShadeType: 2
```

AJ. The Polyline Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvannotate/app-defaults/Polyline`

Inheritance

manager -> graphics -> polyline

Class-Specific Resources

none

Sub-Parts

`The Polyline object has no sub-parts.`

AK. The Position Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvimage/app-defaults/Position`

Inheritance

manager -> graphics -> string -> position

Class-Specific Resources

positionShowValue
positionUpdatemode

Sub-Parts

`The Position object has no sub-parts.`

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_POSITION_SHOW_VALUE
! If TRUE, the position object will display information in
! (X x Y = Z) format, where X and Y display the location of the
! pointer in the data object, and Z is the pixel value at that
! location.  If FALSE, the position object will simply display
! (X x Y) location information;  the pixel value will be omitted.
!
!*Position.positionShowValue: true

! Corresponding attribute: XVW_POSITION_UPDATEMODE
! Whether position object is updated on continuous motion or button press.
! Values include:
!     KPOSITION_UM_CONTINUOUS      0
!     KPOSITION_UM_BUTTON_PRESS    1
!
!*Position.positionUpdatemode: 0
*Position*foreground: black
```

AL. The PrintMapVal Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvimage/app-defaults/PrintMapVal`

Inheritance

manager -> graphics -> color -> printmapval

Class-Specific Resources

printmapvalWidth
printmapvalHeight
printmapvalShowcolor
printmapvalUpdatemode
printmapvalPolicy

Sub-Parts

```
The PrintMapVal object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_PRINTMAPVAL_WIDTH
! The number of map column values that should be displayed in a
! horizontal direction on the printmapval display.
!
!*PrintMapVal.printmapvalWidth: 8

! Corresponding attribute: XVW_PRINTMAPVAL_HEIGHT
! The number of map column values that should be displayed in a
```

```
! vertical direction on the printmapval display.
!
!*PrintMapVal.printmapvalHeight: 8

! Corresponding attribute: XVW_PRINTMAPVAL_SHOWCOLOR
! Whether or not the color of the pixel under the pointer should be
! used as the background color for the label object in which the map values
! are displayed.
!
!*PrintMapVal.printmapvalShowcolor: false

! Corresponding attribute: XVW_PRINTMAPVAL_UPDATEMODE
! Whether printmapval object is updated on pointer motion or button press,
! values include:
!       KPRINTMAPVAL_UM_CONTINUOUS      0
!       KPRINTMAPVAL_UM_BUTTON_PRESS    1
!
!*PrintMapVal.printmapvalUpdatemode: false

! Corresponding attribute: XVW_PRINTMAPVAL_POLICY
! See manual for description, values include:
!       KPRINTMAPVAL_DISPLAYEDVALUES  1
!       KPRINTMAPVAL_MAPDATAVALUES    2
!
!*PrintMapVal.printmapvalPolicy: 1
```

AM. The PrintPixel Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvimage/app-defaults/PrintPixel`

Inheritance

manager -> graphics -> color -> printpixel

Class-Specific Resources

printpixelWidth
printpixelHeight
printpixelShowcolor
printpixelUpdatemode

Sub-Parts

The PrintPixel object has no sub-parts.

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_PRINTPIXEL_WIDTH
! The number of map column values that should be displayed in a
! horizontal direction on the printpixel display.
```

```
!
!*PrintPixel.printpixelWidth: 8

! Corresponding attribute: XVW_PRINTPIXEL_HEIGHT
! The number of map column values that should be displayed in a
! vertical direction on the printpixel display.
!
!*PrintPixel.printpixelHeight: 8

! Corresponding attribute: XVW_PRINTPIXEL_SHOWCOLOR
! Whether or not the color of the pixel under the pointer should be
! used as the background color for the label object in which the map values
! are displayed.
!
!*PrintPixel.printpixelShowcolor: false

! Corresponding attribute: XVW_PRINTPIXEL_UPDATEMODE
! Whether printpixel object is updated on pointer motion or button press,
! values include:
!       KPRINTPIXEL_UM_CONTINUOUS      0
!       KPRINTPIXEL_UM_BUTTON_PRESS    1
!
!*PrintPixel.printpixelUpdatemode: false
```

AN. The Pseudo Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvimage/app-defaults/Pseudo`

Inheritance

manager -> graphics -> color -> pseudo

Class-Specific Resources

pseudoShowPalette
pseudoUpdateOnadd
pseudoUseAlpha

Sub-Parts

```
The sub-parts of the Pseudo object include:
*Pseudo.palette   /* palette object,
                     display color palette for psuedo object          */
*Pseudo.red       /* integer object,
                     allows user to specify red value                 */
*Pseudo.green     /* integer object,
                     allows user to specify green value               */
*Pseudo.blue      /* integer object,
                     allows user to specify blue value                */
*Pseudo.alpha     /* integer object,
                     allows user to specify alpha channel             */
You may specify resources for sub-parts of the pseudo object as desired.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_PSEUDO_SHOW_PALETTE
! Show the palette?
!
!*Pseudo.pseudoShowPalette: true

!
! Corresponding attribute: XVW_PSEUDO_PALETTE_TYPE
!                          (Setting resource on *Pseudo.palette sub-part)
! Specify palette type as one of:
! KPALETTE_TYPE_COLORBAR   1
! KPALETTE_TYPE_COLORCELL  2
! KPALETTE_TYPE_COLORWHEEL 3
!
!*Pseudo.palette.paletteType: 1
!

!
! Corresponding attribute: XVW_PSEUDO_UPDATE_ONADD
! Should pixels immediately change to the color specified by R,G,B
! when they are added to the pseudocolor list?
!
!*Pseudo.pseudoUpdateOnadd: false

!
! Corresponding attribute: XVW_PSEUDO_USE_ALPHA
! Provide integer object to change alpha channel?
!
!*Pseudo.pseudoUseAlpha: false
```

AO. The Rectangle Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvannotate/app-defaults/Rectangle`

Inheritance

manager -> graphics -> rectangle

Class-Specific Resources

none

Sub-Parts

```
The Rectangle object has no sub-parts.
```

AP. The RootWindow Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/RootWindow`

Inheritance

manager -> rootwindow

Class-Specific Resources

none

Sub-Parts

```
The RootWindow object has no sub-parts.
```

AQ. The RowCol Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/RowCol`

Inheritance

manager -> rowcol

Class-Specific Resources

rowcolNumberAcross
rowcolSpacing

Sub-Parts

```
The RowCol object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding Attribute: XVW_ROWCOL_NUMBER_ACROSS
! Number of child objects to lay out in one row before a new row is started.
!
!*RowCol.rowcolNumberAcross: 1

! Corresponding Attribute: XVW_ROWCOL_SPACING
! Specifies minimum spacing between each object (see XVW_ROWCOL_SPACING).
!
!*RowCol.rowcolSpacing: -1
```

AR. The String Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvannotate/app-defaults/String`

Inheritance

manager -> graphics -> string

Class-Specific Resources

stringJustification

Sub-Parts

```
The String object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_STRING_JUSTIFICATION
! Justification of string, one of:
!
!       KSTRING_JUSTIFY_CENTER        1
!       KSTRING_JUSTIFY_TOP           2
!       KSTRING_JUSTIFY_BOTTOM        3
!       KSTRING_JUSTIFY_LEFT          4
!       KSTRING_JUSTIFY_RIGHT         5
!       KSTRING_JUSTIFY_TOPRIGHT      6
!       KSTRING_JUSTIFY_TOPLEFT       7
!       KSTRING_JUSTIFY_BOTTOMRIGHT   8
!       KSTRING_JUSTIFY_BOTTOMLEFT    9
!
!*String.stringJustification: 4
```

AS. The StringValue Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvannotate/app-defaults/StringValue`

Inheritance

manager -> graphics -> string -> stringvalue

Class-Specific Resources

stringFormat

Sub-Parts

```
The StringValue object has no sub-parts.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_STRING_JUSTIFICATION
!                          (setting inherited resources)
! Justification of stringvalue, one of:
!       KSTRING_JUSTIFY_CENTER        1
!       KSTRING_JUSTIFY_TOP           2
!       KSTRING_JUSTIFY_BOTTOM        3
!       KSTRING_JUSTIFY_LEFT          4
!       KSTRING_JUSTIFY_RIGHT         5
!       KSTRING_JUSTIFY_TOPRIGHT      6
!       KSTRING_JUSTIFY_TOPLEFT       7
!       KSTRING_JUSTIFY_BOTTOMRIGHT   8
!       KSTRING_JUSTIFY_BOTTOMLEFT    9
!
!*StringValue.stringJustification: 4

!
! Corresponding attribute: XVW_STRING_EMPHASIZE
!                          (setting inherited resources)
! Emphasize stringvalue?
!
*StringValue.stringEmphasize: true

!
! Corresponding attribute: XVW_STRING_FORMAT
! Format of number displayed by stringvalue object
!
!*StringValue.stringFormat: "%g"
```

AT. The TextDisplay Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/TextDisplay`

Inheritance

manager -> viewport -> textdisplay

Class-Specific Resources

textdisplayRomanFontname
textdisplayBoldFontname
textdisplayItalicFontname
textdisplayHelveticaFontname
textdisplaySymbolFontname

textdisplayRoff
textdisplayIndent
textdisplayWordcolor
textdisplayHightlightcolor

Sub-Parts

```
Sub-parts of the TextDisplay object include:
*TextDisplay.notifywindow   /* notifywindow object,
                               notifies user when new page is formatting */
```

Resource Specifications in App-defaults File

```
! Corresponding Attribute: XVW_TEXTDISPLAY_ROMAN_FONTNAME
! Font to be used for "normal" text
!
!*TextDisplay.textdisplayRomanFontname: fixed

!
! Corresponding Attribute: XVW_TEXTDISPLAY_BOLD_FONTNAME
! Font to be used for bold text
!
!*TextDisplay.textdisplayBoldFontname:*-new century schoolbook-bold-r-normal--12-120-

!
! Corresponding Attribute: XVW_TEXTDISPLAY_ITALIC_FONTNAME
! Font to be used for italic text
!
!*TextDisplay.textdisplayItalicFontname:*-new century schoolbook-bold-i-normal--12-120

!
! Corresponding Attribute: XVW_TEXTDISPLAY_HELVETICA_FONTNAME
! Font to be used for helvetica text
!
!*TextDisplay.textdisplayHelveticaFontname:*-adobe-helvetica-medium-r-normal--12-120-

!
! Corresponding Attribute: XVW_TEXTDISPLAY_SYMBOL_FONTNAME
! Font to be used for symbol text
!
!*TextDisplay.textdisplaySymbolFontname:*-symbol-medium-r-normal--*-120-*

!
! Corresponding Attribute: XVW_TEXTDISPLAY_ROFF
! Format roff commands?
!
!*TextDisplay.textdisplayRoff: true

!
! Corresponding Attribute: XVW_TEXTDISPLAY_INDENT
! Number of pixels to indent before printing text
!
!*TextDisplay.textdisplayIndent: 10
!
! Corresponding Attribute: XVW_TEXTDISPLAY_WORDCOLOR
! Color to use with "hot" words
!
```

```
!*TextDisplay.textdisplayWordcolor: #0000ff

!
! Corresponding Attribute: XVW_TEXTDISPLAY_HIGHLIGHTCOLOR
! Color to use with highlighted words
!
!*TextDisplay.textdisplayHightlightcolor: #ff0000
```

AU. The TextInput Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/TextInput`

Inheritance

manager -> textinput

Class-Specific Resources

none

Sub-Parts

```
Sub-parts of the TextInput object include:
*TextInput.cr_pixmap /* pixmap object,
                        represents a "live" selection              */
*TextInput.text      /* text object,
                        parameter box in which user enters value   */
*TextInput.label     /* labelstring object,
                        displays label of textinput selection      */
You may specify resources for sub-parts of the textinput object as desired.
```

Resource Specifications in App-defaults File

```
! Corresponding Attribute: XVW_PIXMAP_FILENAME
!                          (Setting resources on *TextInput.cr_pixmap sub-part)
! Specify pixmap to appear in upper left hand corner of textinput object
!
!*TextInput.cr_pixmap.pixmapFilename: pixmaps:lightning


!
! Corresponding Attribute: XVW_FOREGROUND, XVW_BACKGROUND
!                          (Setting resources on *TextInput.label sub-part)
! Specify foreground color & font for label of textinput object
!
!*TextInput.label.foreground: #000000
!*TextInput.label.background: #ffffff

!
! Corresponding Attribute: XVW_FOREGROUND, XVW_BACKGROUND, XVW_FONTNAME
```

```
!                          (Setting resources on *TextInput.text sub-part)
! Specify foreground, background color & font for button of textinput object
!
!*TextInput.text.foreground: #000000
!*TextInput.text.background: #979797
!*TextInput.text.fontName:       fixed
!*TextInput.text.fontList:   fixed
```

AV. The TextString Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvisual/app-defaults/TextString`

Inheritance

manager -> graphics -> string -> textstring

Class-Specific Resources

textstringInsertion

Sub-Parts

The TextString object has no sub-parts.

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_TEXTSTRING_INSERTION
! The position at which the user will be inserting text.
! See manual for details.
!
!*TextString.textstringInsertion: 0

!
! Corresponding attributes: XVW_FOREGROUND, XVW_BACKGROUND
!                           (setting inherited resources)
! Foreground & Background colors of TextString object
!
*TextString.foreground:  yellow
*TextString.background:  black
```

AW. The Threshold Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvimage/app-defaults/Threshold`

Inheritance

manager -> graphics -> color -> threshold

Class-Specific Resources

thresholdShowPalette
thresholdPolicy
thresholdClipPixelVal
thresholdClipAccept
thresholdThresPixelVal
thresholdThresInvert

Sub-Parts

```
The sub-parts of the Threshold object include:
*Threshold.palette   /* palette object,
                     displays color palette for threshold object         */
*Threshold.lower     /* integer object,
                     allows user to specify lower range value            */
*Threshold.upper     /* integer object,
                     allows user to specify upper range value            */
*Threshold.range     /* integer object,
                     allows user to change upper & lower values together */
You may specify resources for sub-parts of the threshold object as desired.
```

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_THRESHOLD_SHOW_PALETTE
! Show the palette?
!
!*Threshold.thresholdShowPalette: true

!
! Corresponding attribute: XVW_PALETTE_TYPE
!                          (Setting resource on *Threshold.palette sub-part)
! Specify palette type as one of:
! KPALETTE_TYPE_COLORBAR   1
! KPALETTE_TYPE_COLORCELL  2
! KPALETTE_TYPE_COLORWHEEL 3
!
!*Threshold.palette.paletteType: 1
!

!
! Corresponding attribute: XVW_THRESHOLD_POLICY
! See manual for explanation; values include:
!        KTHRESHOLD_POLICY_CLIP   1
!        KTHRESHOLD_POLICY_THRESH 2
!
!*Threshold.thresholdPolicy: 1

!
! Corresponding attribute: XVW_THRESHOLD_CLIP_PIXELVAL
```

```
!
!*Threshold.thresholdClipPixelVal: 0

!
! Corresponding attribute: XVW_THRESHOLD_CLIP_ACCEPT
!
!*Threshold.thresholdClipAccept: true

!
! Corresponding attribute: XVW_THRESHOLD_THRESH_PIXELVAL
!
!*Threshold.thresholdThreshPixelVal: 255

!
! Corresponding attribute: XVW_THRESHOLD_THRES_INVERT
!
!*Threshold.thresholdThresInvert: false
```

AX. The Timer Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvannotate/app-defaults/Timer`

Inheritance

manager -> graphics -> string -> stringvalue -> timer

Class-Specific Resources

timerUpdatetime

Sub-Parts

The Timer object has no sub-parts.

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_TIMER_UPDATETIME
! How often the timer is updated, in fractions of a second.
!
!*Timer.timerUpdatetime: 0.1
```

AY. The Viewport Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/Viewport`

Inheritance

manager -> viewport

Class-Specific Resources

vpAllowHoriz
vpAllowVert
vpForceHoriz
vpForceVert
vpUseBottom
vpUseRight
vpXoffset
vpYoffset

Sub-Parts

```
Sub-parts of the Viewport object include:
*Viewport.clip      /* manager object,
                       imposes physical (clip) size of viewport area */
*Viewport.plane     /* manager object,
                       makes up full (plane) size of viewport contents */
*Viewport.vertical  /* scrollbar object,
                       the vertical scrollbar */
*Viewport.horizontal /* scrollbar object,
                        the horizontal scrollbar */
```

Resource Specifications in App-defaults File

```
! Corresponding Attribute: XVW_VP_ALLOW_HORIZ
! Allow horizontal scrollbar?
!
!*Viewport.vpAllowHoriz: true

!
! Corresponding Attribute: XVW_VP_ALLOW_VERT
! Allow vertical scrollbar?
!
!*Viewport.vpAllowVert:  true

!
! Corresponding Attribute: XVW_VP_FORCE_HORIZ
! Insist on horizontal scrollbar (whether or not it is needed)?
!
!*Viewport.vpForceHoriz: true

!
! Corresponding Attribute: XVW_VP_FORCE_VERT
! Insist on vertical scrollbar (whether or not it is needed)?
!
!*Viewport.vpForceVert: true

!
! Corresponding Attribute: XVW_VP_USE_BOTTOM
```

```
! Horizontal scrollbar appearing at bottom?
!
!*Viewport.vpUseBottom: true

!
! Corresponding Attribute: XVW_VP_USE_RIGHT
! Vertical scrollbar appearing at right?
!
!*Viewport.vpUseRight: true

!
! Corresponding Attribute: XVW_VP_XOFFSET
! The offset of the viewport in pixels from the left side
!
!*Viewport.vpXoffset: 0

!
! Corresponding Attribute: XVW_VP_YOFFSET
! The offset of the viewport in pixels from the top
!
!*Viewport.vpYoffset: 0
```

AZ. The Warn Object

Generated From App-defaults file:

`$DESIGN/objects/library/xvobjects/app-defaults/Warn`

Inheritance

manager -> warn

Class-Specific Resources

warnPixmapfile

Sub-Parts

```
Sub-parts of the Warn object include:
*Warn.pixmap /* pixmap object,
                decorates upper left hand corner of Warn object */
*Warn.text   /* text object,
                displays warning message                        */
*Warn.button /* button object,
                acknowledgement button                          */
*Warn.label  /* labelstring object,
                label at top of warn object                     */
You may specify resources for sub-parts of the warn object as desired.
```

Resource Specifications in App-defaults File

```
! Corresponding Attribute: XVW_WARN_PIXMAPFILE
! Specify pixmap to appear in upper left hand corner of warn object
!
!*Warn*warnPixmapfile: $DESIGN/objects/library/xvobjects/pixmaps/warn.xpm


!
! Corresponding Attribute: XVW_FOREGROUND, XVW_FONTNAME
!                          (Setting resources on *Warn.label sub-part)
! Specify foreground color & font for label of warn object
!
!*Warn.label.foreground: #000000
!*Warn.label.fontName:       fixed

!
! Corresponding Attribute: XVW_FOREGROUND, XVW_BACKGROUND, XVW_FONTNAME
!                          (Setting resources on *Warn.button sub-part)
! Specify foreground, background color & font for button of warn object
!
!*Warn.button.foreground: #000000
!*Warn.button.background: #ffff00
!*Warn.button.fontName:       fixed
!*Warn.button.fontList:   fixed


!
! Corresponding Attribute: XVW_FOREGROUND, XVW_FONTNAME
!                          (Setting resources on *Warn.text sub-part)
! Specify foreground, background color & font for text of warn object
!
!*Warn.text.foreground: #000000
!*Warn.text.fontName:       fixed
!*Warn.text.fontList:   fixed
```

BA. The Zoom Object

Generated From App-defaults file:

`$ENVISION/objects/library/xvimage/app-defaults/Zoom`

Inheritance

manager -> graphics -> color -> image -> zoom

Class-Specific Resources

zoomFactor
zoomLocationmarker
zoomUpdatemode
zoomInterpolate

Sub-Parts

The Zoom object has no sub-parts.

Resource Specifications in App-defaults File

```
! Corresponding attribute: XVW_ZOOM_FACTOR
! The zoom factor; see manual for explanation.
!
!*Zoom.zoomFactor: 1.0

!
! Corresponding attribute: XVW_ZOOM_LOCATIONMARKER
! Type of cursor used to mark center of zoom window; values include:
!       KZOOM_LM_NONE           0
!       KZOOM_LM_CROSS          1
!       KZOOM_LM_BOX            2
!       KZOOM_LM_DOT            3
!*Zoom.zoomLocationmarker: 1

!
! Corresponding attribute: XVW_ZOOM_UPDATEMODE
! Whether zoom is updated on pointer motion or button press, values include:
!       KZOOM_UM_CONTINUOUS     0
!       KZOOM_UM_BUTTON_PRESS   1
!
!*Zoom.zoomUpdatemode: 0

!
! Corresponding attribute: XVW_ZOOM_INTERPOLATE
! Type of interpolation to be used; currently, only KZERO_ORDER (2)
! (pixel replication) is supported.
!
!*Zoom.zoomInterpolate: 2
```

Table of Contents