SlingShot Extensions to XView

Programmer's Handbook for Version 2.1

Brian Warkentine

October 25, 1993

Preface

The XView toolkit provides a number of user interface objects such as Frames, Icons, and Panel items. While these are necessary and appropriate for many types applications, programmers are forced to use Xlib to implement other types of user interfaces. Unfortunately, Xlib programming is difficult and tedious. The SlingShot extensions provide one answer to this dilemma, by supplying a set of objects and event handling routines that implement commonly needed functionality.

The SlingShot extensions to XView are based on the XView toolkit. Programmers who already know XView will find the SlingShot extensions very familiar because the XView API (*xv_create()*, *xv_set()*, *xv_get()*, etc.) is used.

This document describes release 2.1 of SlingShot extensions. Release 2.1 is based on XView 3.0 (or later) and incorporates the XView 3.0 selections and drag and drop interfaces. This document is intended as a guide to application developers who want to write applications using the SlingShot extensions. Readers should have an understanding of XView programming.

Introduction

Like many modern user interface toolkits, object oriented classes can be added to XView. The SlingShot extensions take advantage of this feature by adding 17 new object types. The SlingShot extension hierarchy and its relationship to the XView hierarchy is shown in Figure 1.

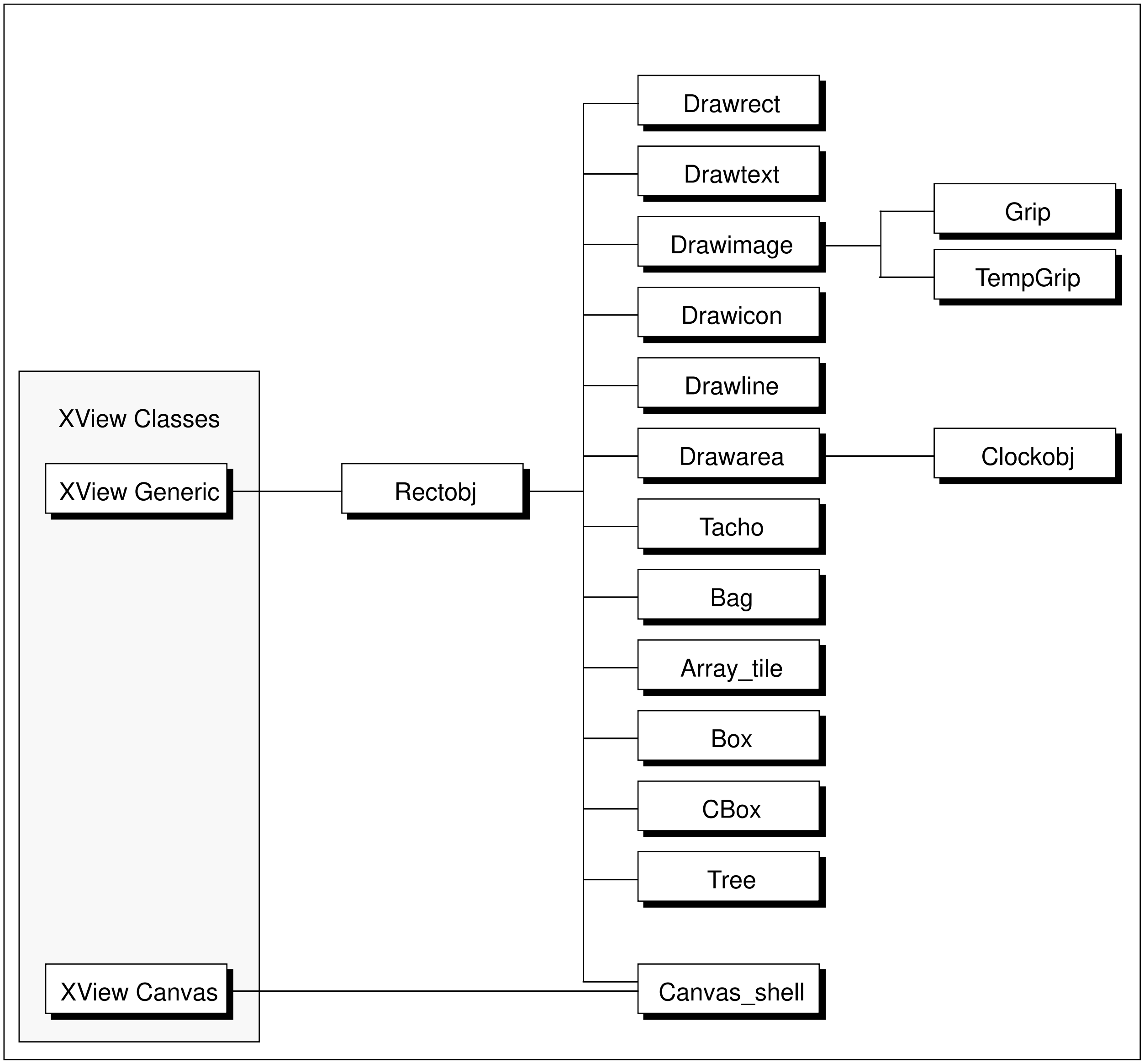

Figure 1: The SlingShot extension hierarchy

In Figure 1, the two classes in the box on the left are part of XView. The remaining classes are part of the SlingShot extensions. The Rectobj class (extension) provides the base functionality, such as support for common attributes, and provides hooks for repainting, event handling, and geometry management. The remaining subclasses of the rectobj use the rectobj infrastructure and define additional functionality. Also note that the Canvas_shell is actually subclassed from the XView Canvas but shares some functionality implemented by the Rectobj.

A Sample Program

To illustrate the features of the SlingShot extensions, let's look at a simple program. (This example is included in the source distribution as example1.c. Additional examples are also included with the source distribution.)

```
#include <xview/frame.h>
#include <sspkg/canshell.h>
#include <sspkg/rectobj.h>
#include <sspkg/drawobj.h>

main(argc, argv)
             int          argc;
             char         *argv[];
{
             Frame        frame;
             Canvas_shell shell;
             Rectobj      rectobj1;
             Drawrect     drawrect1;
             Drawtext     drawtext1;

             xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);

             frame = (Frame) xv_create(NULL, FRAME,
                          FRAME_LABEL, argv[0],
                          NULL);

             shell = (Canvas_shell) xv_create(frame, CANVAS_SHELL, NULL);

             rectobj1 = (Rectobj) xv_create(shell, BAG,
                          XV_X, 100,
                          NULL);

             drawrect1 = (Drawrect) xv_create(rectobj1, DRAWRECT,
                          XV_Y, 16,
                          XV_WIDTH, 200,
                          XV_HEIGHT, 200,
                          NULL);

             drawtext1 = (Drawtext) xv_create(rectobj1, DRAWTEXT,
                          DRAWTEXT_STRING, "Hello World",
                          XV_X, 60,
                          XV_Y, 10,
                          NULL);

             xv_main_loop(frame);
}
```

Figure 2: A sample program

The example program in Figure 2 creates four SlingShot objects and the running program is shown in Figure 3. The first SlingShot object, *shell*, is the window that the other objects will appear in. Its type is `CANVAS_SHELL`, a subclass of the XView `CANVAS` that knows about the other SlingShot objects that appear within it. In fact, all other SlingShot objects are "windowless" and are only visible when inside a `CANVAS_SHELL`. You can create a SlingShot `CANVAS_SHELL` whereever you would ordinarily create or use a XView `CANVAS`.

The second object that is created, *rectobj1*, is a `BAG`. This is an invisible object that serves as a place to put other objects[1]. The `BAG` is a class that shrinks or expands to fit the dimensions of other objects. In this example, the `BAG` object is used to hold two objects, *drawrect1* and *drawtext1*, which appear as a rectangle and text.

Like other XView objects, SlingShot objects are organized into hierarchies. Each object has an owner and can have several *children*. In the example, *rectobj1* is a subtree or *child* of *shell*, *drawtext1* and *drawrect1* are children of *rectobj1*. The parent-child relationship is usually established by the first argument to *xv_create()*.

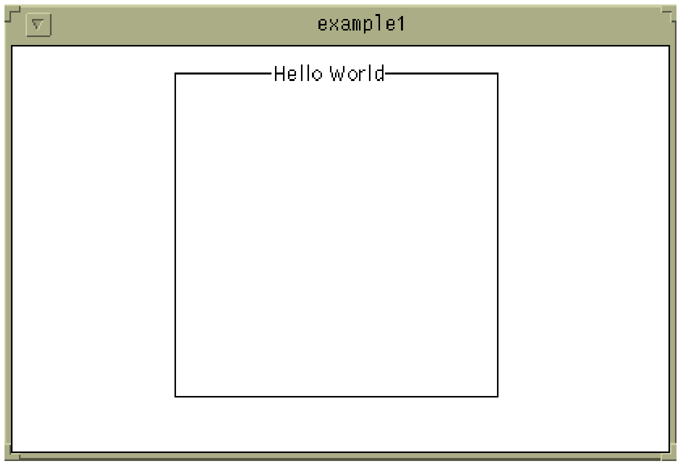

Figure 3: Execution of the sample program

The position and sizes of SlingShot objects are controlled with the `XV_X`, `XV_Y`, `XV_WIDTH` and `XV_HEIGHT` attributes. The positions are relative to the object's owner. In the example, *rectobj1* is positioned at (100, 0) of the *shell*; *drawtext1* is positioned at (10, 60) of *rectobj1*; *drawrect1* is positioned at (0, 16) of *rectobj1*. The width and height of *drawrect1* are set to 200. Note that the width and height of *drawtext1* do not need to be set, because it is calculated by the dimensions of the text within the object.

Whenever objects are nested, as the *drawrect1* and *drawtext1* are within *rectobj1*, the owner is involved in the positioning of its children. In the default case, as with the `BAG` and some other rectobj subclasses, the `XV_X` and `XV_Y` attributes of the child are controlled by the application. Other classes, such as the `TREE` and `ARRAY_TILE`, explicitly position their children according to constraints specified with other attributes. This positioning is called geometry management.

Most SlingShot objects are rectangular. All provide `XV_X`, `XV_Y`, `XV_WIDTH`, and `XV_HEIGHT` attributes, but for some classes these only describe the bounding rectangle. For example, a `DRAWLINE` object, which is a line between two points cannot be described with rectangular dimensions alone.

Objects may overlap and the changes beneath an object do not affect the object on the top. Because *drawtext1* was created after *drawrect1* and their bounding rectangles overlap, *drawtext1* appears on top and obscures part of *drawrect1*. If the color of *drawrect1* is changed later, it does not scribble on top of *drawtext1*.

Some SlingShot objects are selectable. If you click the mouse on the "Hello World" in this example, it will become highlighted. The SlingShot extensions handle these events for you and provide callback functions to let you know what has happened. For example, when a user double clicks on an object, a function specified by the application with `RECTOBJ_DOUBLE_CLICK_PROC` is called.

1. There is a built in debugging tool that lets you view the boundaries of all the objects in a canvas_shell. Invoke this by holding down the keyboard control and shift buttons and clicking the left mouse button.

Handling Events

In contrast to the XView Canvas, the SlingShot extensions provide higher level events. The low level event stream from the window serveris processed into higher level abstractions such as single click, double click, and drag and drop.

Help and Menus

Each SlingShot object can have help information assigned to it, and can share help with its children. An application associates help information with an object by setting the `XV_HELP_DATA` attribute (as described in the XView Programming Manual.) When the help button is pressed, the SlingShot extensions find the help information, and then display it with *xv_help_show()*. The SlingShot extensions find the help data by exploiting the hierarchial relationships between objects. If there is no `XV_HELP_DATA` attached to the object that the help event occurred on, then the object's owners are searched in turn until help data is found or until the canvas_shell is reached. This makes it easy to specify help information for a group of objects.

Each SlingShot object can also have a menu assigned to it, and can share the menu with its children. An application first creates a menu, then associates the menu with an object by setting the `RECTOBJ_MENU` attribute. When the right mouse button is pressed, the SlingShot extensions find the menu and display it with XView's *menu_show()*. Menus are searched for among the object's owners using the object hierarchy, in the same manner that help data is searched for.

In Figure 4, a menu called *my_menu* is attached to *rectobj1* using `RECTOBJ_MENU`, and help data is associated using `XV_HELP_DATA`. It is assumed that *my_menu* was previously created.

```
Menu          my_menu;
Rectobj       rectobj1;

...

xv_set(rectobj1,
       RECTOBJ_MENU, my_menu,
       XV_HELP_DATA, "sample_program:filebutton",
       NULL);
```

Figure 4: Assigning help and menus to an object

Single Click, Double Click

Single click and double click callbacks are provided by the SlingShot extensions. When a user clicks on a object such as the *drawtext1* object in Figure 2, it becomes highlighted. When the user releases the mouse button, the single click callback function is called. If the user clicks on the object again within a certain time limit, the double click callback function is called.

Figure 5 shows how these callbacks are specified with `RECTOBJ_SINGLE_CLICK_PROC` and `RECTOBJ_DOUBLE_CLICK_PROC`. On the first click, *my_single_click_proc()* will be called. On the double click, *my_double_click_proc()* will be called. The arguments to the callback provide context for the click operation: The *paint_window* and *canvas_shell* are the windows that *event* occurred on, and *click_object* is the SlingShot object that the user's mouse was positioned over.

```
void my_single_click_proc(
      Paint_window        paint_window,
      Event               *event,
      Canvas_shell        canvas_shell,
      Rectobj             click_object)
{
      /* user single clicked on click_object, do something */
}

void my_dbl_click_proc(
      Paint_window        paint_window,
      Event               *event,
      Canvas_shell        canvas_shell,
      Rectobj             click_object)
{
      /* user double clicked on click_object, do something */
}

...

      xv_set(drawtext1,
            RECTOBJ_SINGLE_CLICK_PROC, my_single_click_proc,
            RECTOBJ_DBL_CLICK_PROC,    my_dbl_click_proc,
            NULL);
```

Figure 5: Setting up single and double click callbacks on an object

Low Level Event Handling

As mentioned, all the SlingShot extensions except the Canvas_shell are *windowless*. Because X and XView send events to windows only, the windowless SlingShot objects have to cooperate to handle events.

Events that have a location, such as a mouse click event, need to be associated with the object at that location. Because objects can overlap, this needs to be done carefully. An event is *mapped* to a single object if the x and y location of the event is within the bounds of the SlingShot object. Because some objects are non-rectangular, a callback function (`RECTOBJ_MAP_EVENT_PROC`) actually implements the bounds checking.

Once the event is "mapped" to an object, a callback function for that object (`RECTOBJ_EVENT_PROC`) is invoked. As you might imagine, the `RECTOBJ_EVENT_PROC` is a low level interface so most SlingShot classes set up default event handlers to transform events into higher level callbacks, such as the single and double click callbacks. The prebuilt event handlers available for application use are:

- *rectobj_selection_event_proc()*

  This function supports selectable objects and invokes callbacks for selections, single click, double click, and start drag.

- *rectobj_button_event_proc()*

  This function makes the object act like a button: the object is highlighted while the mouse button is pressed over the object, and unhighlighted in normal conditions. It calls the `RECTOBJ_SINGLE_CLICK_PROC` when the user releases the button.

- *rectobj_toggle_event_proc()*

  This function toggles an object between two states, highlighted and normal.

- *rectobj_background_event_proc()*

  Supports rubberbanding and selection of objects that fit in the rubberbanded area.

All the event handlers support callbacks for handling Drag & Drop drop events, and XView help and menus are displayed autonomously.

Many of the objects use *rectobj_selection_event_proc()* by default because it supports the most general set of features. The application is free to set any of these functions for their `RECTOBJ_EVENT_PROC` depending on the behavior that is desired.

There are alternatives if none of these event functions fit your needs. One option is to write your own event proc and assign it with `RECTOBJ_EVENT_PROC`. The advantage of this callback is that it only gets called when events occur on the object. The other option is to use *notify_interpose_event_func()* to catch events before they arrive at the event proc of the canvas_shell's paint windows. This option could require more work, but may get the effect you desire.

Selections

The SlingShot extensions provide for highlighting and unhighlighting of an object when the user clicks on or rubberbands around the object. The list of selected objects is accessible by calling *rectobj_get_selected_list()*. This is in itself not enough to interact with other desktop applications. The application has to do a little more work to interoperate with the XView selection interface. This can only be done by the application because the application knows about what the selected objects represent.

To use the XView selection interface, the application should create an XView Selection_owner object and set appropriate attributes, such as the conversion and done functions. It may wish to create a Selection_item also. The SlingShot extensions will make use of a Selection_owner object if it is informed of its existence. This is done by setting `RECTOBJ_SELECTION_OWNER`. Once a Selection_owner is specified for an object, the SlingShot extensions can automatically set the `SEL_OWN`, and `SEL_TIME` attributes to acquire the primary selection when the user highlights an object. The SlingShot extensions will also handle the loss of a selection and will unhighlight the object. (It does not use the `SEL_LOSE_PROC` for this, so you don't have to worry about it overriding the application callback.)

`RECTOBJ_SELECTION_OWNER` may be set on each object that you want to have highlighted, or on a group of objects. The rectobj hierarchy is searched upward until a selection owner is found. Only one primary selection can be owned at a time, so if every object has a different owner, simultaneous selected objects will not be possible.

There is an example of interacting with selections in the examples directory of the source distribution.

Initiating Drag and Drop

To initiate drag and drop operations, the application should have set up the selection interface by creating a `DRAGDROP` and associated objects. To get notification of when to begin processing the drag and drop operation, set an object's `RECTOBJ_START_DRAG_PROC`. This callback is invoked when a user clicks on an object and drags it a few pixels. Within this callback, the drag and drop operation can be started by calling *dnd_send_drop()*. `RECTOBJ_DRAGGABLE` should be set to true to enable this callback.

Receiving Drag and Drops

Several attributes control how drag preview and drop events are processed. `CANVAS_SHELL_AUTO_DROP_SITE` tells the canvas_shell to create and maintain a `DROP_SITE_ITEM`. The drop site area is the size of the paint window and one is created for every paint window. When `RECTOBJ_ACCEPTS_DROP` is `TRUE`, the SlingShot object is notified of drag events (inside it's dimensions) and of drop events by the `RECTOBJ_DROP_PROC`. This callback should look at the id of the event for `ACTION_DRAG_PREVIEW`, `ACTION_DRAG_COPY`, etc., and do the appropriate action, such as highlighting the object or calling *dnd_decode_drop()*.

Examples of handling drag and drop with the SlingShot extensions can be found in the examples directory of the source distribution.

Lists

Doubly linked lists are used in the SlingShot extensions for several purposes. For example, the children of a SlingShot object are linked up using a linked list and the function *rectobj_get_selected_list()* returns a linked list of SlingShot objects that are selected. In some cases, applications will need to use these list functions.

The example in Figure 6 illustrates iterating through a list of objects. Some of the list operations used here are macros, as implemented in *<sspkg/list.h>*. This example uses the *list_for()* macro to interate through the list of children of another SlingShot object, *a_parent_rectobj*.

```
Rectobj_list  *my_list;
Rectobj       child_rectobj;

my_list = (Rectobj_list*) xv_get(a_parent_rectobj, RECTOBJ_CHILDREN);

/* iterate through all the children */
list_for(my_list) {
        /* get the child out of the list node */
        child_rectobj = RECTOBJ_LIST_HANDLE(my_list);
        /* do something with it */
        xv_set(child_rectobj,
                RECTOBJ_FG, orange,
                NULL);
}
```

Figure 6: Iterating through a list of SlingShot objects

Color

To use color with the SlingShot extensions, the application needs to create an XView cms and assign the `WIN_CMS` attribute of the window (Canvas_shell or Frame). Each object can refer to the cms colors with the `RECTOBJ_FG` and `RECTOBJ_BG` attributes. These attributes are indexes into the cms for the window that they are in. The default color is set by setting the attribute to `-1`.

The default colors for all objects in a canvas_shell can be set by setting the `RECTOBJ_FG` and `RECTOBJ_BG` on the canvas_shell. By default these are the last and first entries of the cms.

Where to get information and sources

The first thing to do when you have a problem or question is to peruse the examples directory. This handbook is admittedly brief, and the examples should prove to be illustrative in areas that are not covered in detail here. (Also note the brief descriptions of the example programs in Appendix E.) A second place for information is Usenet. The Usenet newsgroups comp.windows.open-look and alt.toolkits.xview have a number of readers who are knowledgeable about XView and the SlingShot extensions.

As of this writing, the latest major release is available via ftp on *ftp.x.org* as /contrib/SlingShot2.1.tar.Z. This ftp server is mirrored to various sites around the world, so check your local ftp server first. Previously, an automated mail archive server was set up, but this service has been discontinued.