13
Cursors

A *cursor* is an image that tracks the mouse on the display. Each window has its own cursor which you can change. There are some cursors defined by OPEN LOOK that correspond to specific window manager operations such as resizing or dragging windows. For these cases, you cannot redefine a cursor. However, for windows in your application, you can assign any cursor image you like.

13.1 Creating Cursors

To use the `CURSOR` package, include the header file <*xview/cursor.h*>. It provides the necessary types and definitions for using the package. The cursor object's type is `Xv_Cursor`. Figure 13-1 shows the class hierarchy for a cursor object.

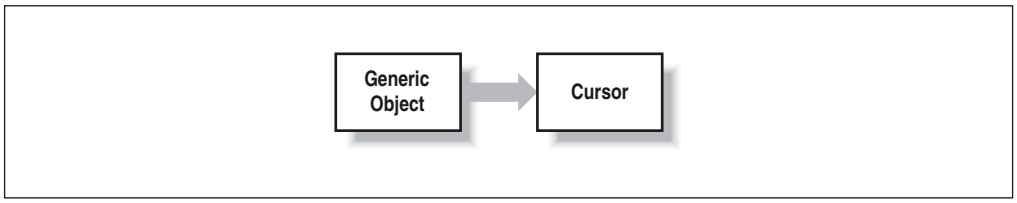

Figure 13-1. Cursor class hierarchy

In general, to create a cursor, create an image and a cursor using that image as the `CURSOR_IMAGE` data:

```
Server_image  svr_image;
Xv_Cursor     cursor;

cursor = (Xv_Cursor)xv_create(owner, CURSOR,
    CURSOR_IMAGE,  svr_image,
    NULL);
```

The `owner` of the cursor may be any XView object. The root window associated with the XView object is used internally by the `CURSOR` package. If `NULL`, then the root window of the default screen is used.

The cursor is then assigned to a window associated with an XView object such as a frame, canvas, or panel:

```
xv_set(window, WIN_CURSOR, cursor, NULL);
```

You must supply the handle of an XView window in the parent parameter when getting `WIN_CURSOR`. Getting `WIN_CURSOR` on the root window will return `NULL`. It is illegal to assign a cursor to a window if the screens do not match. This is normally not a problem unless you are using multiple displays in your application. In this case, you should be sure to use an XView object that has a common display as the owner for the cursor. In the code line above, *`window`* should be the visible window to the application. For canvases and panels, this should be the *paint window*, not the canvas or panel object itself.* If you assign your own cursor to an openwin object (such as a canvas or panel) and the object has been split (either by the user *splitting views* or by the application), then the application is responsible for assigning the cursor to each new paint window.

13.1.1 simple_cursor.c

To introduce how to use the `CURSOR` package, we'll start with a short program that shows how to set the cursor for a canvas.

Example 13-1. The simple_cursor.c program

```
/*
 * simple_cursor.c -- create a cursor (looks like an hourglass) and
 * assign it to a canvas window.
 */
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/cursor.h>
#include <xview/svrimage.h>

/* data that describes the cursor's image -- see SERVER_IMAGE below */
short cursor_bits[ ] = {
/* Width=16, Height=16, Depth=1, */
    0x7FFE,0x4002,0x200C,0x1A38,0x0FF0,0x07E0,0x03C0,0x0180,
    0x0180,0x0240,0x0520,0x0810,0x1108,0x23C4,0x47E2,0x7FFE
};

main(argc, argv)
int argc;
char *argv[ ];
{
    Frame        frame;
    Canvas       canvas;
    Xv_Cursor    cursor;
    Server_image svr_image;

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

*See Chapter 5, *Canvases and Openwin*, for more information about the paint window.

Example 13-1. The simple_cursor.c program (continued)

```
    /*
     * create a server image to use as the cursor's image.
     */
    svr_image = (Server_image)xv_create(XV_NULL, SERVER_IMAGE,
        XV_WIDTH,               16,
        XV_HEIGHT,              16,
        SERVER_IMAGE_BITS,      cursor_bits,
        NULL);
    /*
     * create a cursor based on the image just created
     */
    cursor = (Xv_Cursor)xv_create(XV_NULL, CURSOR,
        CURSOR_IMAGE,           svr_image,
        NULL);

    /*
     * Create a base frame and a canvas
     */
    frame = (Frame)xv_create(XV_NULL, FRAME, NULL);
    canvas = (Canvas)xv_create(frame, CANVAS,
        XV_WIDTH,               100,
        XV_HEIGHT,              100,
        NULL);
    /*
     * set the cursor to the paint window for the canvas
     * Do not set it for the canvas itself.
     */
    xv_set(xv_get(canvas, CANVAS_NTH_PAINT_WINDOW, 0),
        WIN_CURSOR,             cursor,
        NULL);

    window_fit(frame);
    window_main_loop(frame);
}
```

Beware that if a canvas (or any openwin object) is split, the new view (which has a corresponding paint window) does not inherit the cursor from the old view window.* Note that the server images used in cursors must be one-bit deep. Cursors can have two colors associated with them by specifying foreground and background colors; you cannot specify server images whose depths are greater than 1. See Section 13.4, "Color Cursors."

*Chapter 5, *Canvases and Openwin*, discusses splitting views.

13.2 Predefined Cursors

A number of predefined cursors are available in the CURSOR package for use as OPEN LOOK cursors. To use these cursors, you may specify the CURSOR_SRC_CHAR and CURSOR_MASK_CHAR attributes with certain predefined constants as values for these attributes. In <*xview/cursor.h*>, there are some OPEN LOOK cursor defines prefixed by OLC_. When using these attributes, you should not use the CURSOR_IMAGE attribute since you cannot use both simultaneously. Using the previous example, we can remove the SERVER_IMAGE references and modify the call to create the cursor:

```
cursor = xv_create(NULL, CURSOR,
    CURSOR_SRC_CHAR, OLC_BUSY_PTR,
    NULL);
```

Predefined cursors are really images from a pre-built font. The *value* in the attribute-value pair is the character to use from that font—or rather, it is the index into the array of glyphs that the font contains. The glyph from the font is extracted and used as the image. You can use the attribute CURSOR_MASK_CHAR similarly. This image is used as the mask for the source image. If no mask is given, the same image used as the source is used as the mask.*

13.3 The Hotspot and Cursor Location

The *hotspot* on a cursor is the location in which the cursor is located if the user generates an event like pressing a mouse button or typing at the keyboard, or if you were to query its position. For example, if a cursor is shaped like an arrow, the hotspot should be at the tip of the arrow. If the hotspot for a cursor were set to (0, 0), then the hotspot would be the upper-left corner of the image used. A cursor shaped like a bull's eye (16x16) might have its hotspot at (7, 7) to indicate that the focus for the cursor is in the middle.† You set a cursor's hotspot with the attributes CURSOR_XHOT and CURSOR_YHOT. CURSOR_XHOT specifies the *x* coordinate of the hotspot. CURSOR_YHOT specifies the *y* coordinate of the hotspot. You can find out what the current position of the cursor is by using the attribute WIN_MOUSE_XY, as in:

```
r = (Rect *)xv_get(window, WIN_MOUSE_XY);
```

The return value from xv_get() is a pointer to a Rect structure. The r_width and r_height fields of this structure are unused (0, 0), but the r_top and r_left fields indicate the position of the hotspot for the cursor with respect to the window, window. The program in Example 13-2 demonstrates how this is used, and it shows how to create your own pixmap for a cursor image.

*See XCreateGlyphCursor and XCreatePixmapCursor in Volume Two, *Xlib Reference Manual*.
†The value 7, 7 is used because the origin is at 0, 0—not 1, 1.

Example 13-2. The hot_spot.c program

```
/*
 * hot_spot.c -- create a cursor and query its position on the
 * screen and in the panel's window.
 * Our own function, create_cursor(), attaches a new cursor to the
 * window parameter passed into the function.
 */
#include <X11/X.h>
#include <X11/Xlib.h>           /* for the xlib graphics */
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/cursor.h>
#include <xview/svrimage.h>

main(argc, argv)
int argc;
char *argv[ ];
{
    Frame       frame;
    Panel       panel;
    void        do_it();

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

    /*
     * Create a base frame, a panel, and a panel button.
     */
    frame = (Frame)xv_create(XV_NULL, FRAME, NULL);
    panel = (Panel)xv_create(frame, PANEL, NULL);
    create_cursor(xv_get(panel, CANVAS_NTH_PAINT_WINDOW, 0));
    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Push Me",
        PANEL_NOTIFY_PROC,      do_it,
        NULL);

    window_fit(panel);
    window_fit(frame);
    window_main_loop(frame);
}

/*
 * When user selects the panel button, the current mouse location is
 * printed relative to the panel's window and to the screen.
 * This location is governed by the hot spot on the cursor.
 */
void
do_it(item, event)
{
    Rect *r;
    Panel panel = (Panel)xv_get(item, PANEL_PARENT_PANEL);

    r = (Rect *)xv_get(xv_get(panel, XV_ROOT), WIN_MOUSE_XY);
    printf("Root window: %d %d\n", r->r_left, r->r_top);
    r = (Rect *)xv_get(xv_get(panel,
                    CANVAS_NTH_PAINT_WINDOW, 0), WIN_MOUSE_XY);
    printf("Panel window: %d %d\n", r->r_left, r->r_top);
}
```

Example 13-2. The hot_spot.c program (continued)

```
/*
 * create_cursor() creates a bull's eye cursor and assigns it
 * to the window (parameter).
 */
create_cursor(window)
Xv_Window window;
{
    Xv_Cursor      cursor;
    Server_image   image;
    Pixmap         pixmap;
    Display        *dpy = (Display *)xv_get(window, XV_DISPLAY);
    GC             gc;
    XGCValues      gcvalues;

    image = (Server_image)xv_create(XV_NULL, SERVER_IMAGE,
        XV_WIDTH,       16,
        XV_HEIGHT,      16,
        NULL);
    pixmap = (Pixmap)xv_get(image, XV_XID);
    /* Create GC with reversed foreground and background colors to
     * clear pixmap first.  Use 1 and 0 because pixmap is 1-bit deep.
     */
    gcvalues.foreground = 0;
    gcvalues.background = 1;
    gc = XCreateGC(dpy, pixmap, GCForeground|GCBackground, &gcvalues);
    XFillRectangle(dpy, pixmap, gc, 0, 0, 16, 16);
    /*
     * Reset foreground and background values for XDrawArc() routines.
     */
    gcvalues.foreground = 1;
    gcvalues.background = 0;
    XChangeGC(dpy, gc, GCForeground | GCBackground, &gcvalues);
    XDrawArc(dpy, pixmap, gc, 2, 2, 12, 12, 0, 360 * 64);
    XDrawArc(dpy, pixmap, gc, 6, 6, 4, 4, 0, 360 * 64);

    /* Create cursor and assign it to the window (parameter) */
    cursor = xv_create(XV_NULL, CURSOR,
        CURSOR_IMAGE,   image,
        CURSOR_XHOT,    7,
        CURSOR_YHOT,    7,
        NULL);
    xv_set(window, WIN_CURSOR, cursor, NULL);

    /* free the GC -- the cursor and the image must not be freed. */
    XFreeGC(dpy, gc);
}
```

When the program is running, each time the panel button is pushed, it prints the cursor's position relative to the panel's window and relative to the root window (absolute screen coordinates). You can move the base frame around on the screen to see how the root window coordinates change.

The routine `create_cursor()` creates a bull's eye cursor for the window passed as the parameter to the routine. The cursor image must be a `Server_image`, so we first create a server image, then get the `Pixmap` associated with it using the `XV_XID`, and lastly use Xlib graphics to draw two circles in the pixmap.

We need a `GC`, so we create one based on the pixmap obtained from the server image. The pixmap is one-bit deep, so the foreground and background colors are set to 0, 1 (to clear the pixmap first), then to 1, 0 so as to draw the two circles. We then create the cursor using the server image and setting the hotspots accordingly. We free the `gc`, but the `Server_image` (which contains the pixmap) and the `cursor` must not be freed so the cursor can be maintained by the window.

If you would rather set the cursor for a window using raw Xlib calls such as `XCreatePixmapCursor`, `XCreateGlyphCursor`, or `XCreateFontCursor`, use the X window associated with the `Xv_Window` parameter. To get it, use:

```
xv_get(window, XV_XID)
```

and assign an X `Cursor` object to that window.*

13.4 Color Cursors

You can define the foreground and background colors of a cursor independently of the window the cursor is assigned to. You may not have more than two colors per cursor because X does not support color images as cursor glyphs. Thus, to create or modify an existing cursor to have color, you need to specify foreground and background colors. Because of the use of color, the header file *<xview/cms.h>* must be included. The colors are of type `Xv_singlecolor` and should be initialized before use:

```
#include <xview/cms.h>
...
Xv_singlecolor   fg, bg;

bg.red = 250, bg.green = 230, bg.blue = 30;
fg.red = 180, fg.green = 100, fg.blue = 20;

cursor = xv_create(NULL, CURSOR,
    CURSOR_IMAGE,              image,
    CURSOR_FOREGROUND_COLOR,  &fg,
    CURSOR_BACKGROUND_COLOR,  &bg,
    NULL);
```

Note, by default, a cursor is created with a mask equal to the image; therefore, there is no background color. To use a background color, set the cursor's `CURSOR_BACKGROUND_COLOR` and the `CURSOR_IMAGE` attributes, and then set the `CURSOR_OP` as follows:

```
xv_set(cursor,CURSOR_OP,PIX_SRC,NULL);
```

*See Volume One, *Xlib Programming Manual*.

13.5 Support for Text Drag and Drop

The Cursor package supports drag and drop cursors for text. There are attributes that change the cursor to indicate that text is being dragged. When the currently selected item is over an acceptable drop site, a preview cursor is displayed. There is also an attribute to support a reject cursor as well, but it is an unsupported attribute.

The attribute `CURSOR_DRAG_STATE` indicates whether the cursor is over a neutral zone (`CURSOR_NEUTRAL`), a valid drop zone (`CURSOR_ACCEPT`), or an invalid drop zone (`CURSOR_REJECT`). The shape of the cursor varies depending on the state.*

`CURSOR_DRAG_TYPE` changes the cursor to indicate whether a move (`CURSOR_MOVE`) or copy (`CURSOR_DUPLICATE`) operation is being performed. The duplicate version has a shadow. When combined with `CURSOR_STRING`, you get either a text move or a text duplicate cursor.

`CURSOR_STRING` creates a drag and drop Cursor for text. The value of the attribute is the string which is to be displayed inside the *flying punch card*. If the string exceeds 3 characters, only the first 3 characters are displayed, and an arrow is shown within the cursor. `CURSOR_STRING` is mutually exclusive of `CURSOR_IMAGE`, `CURSOR_SRC_CHAR`, and `CURSOR_MASK_CHAR`. The string is not copied. Once the Drag and Drop operation is complete, the objects used in the operation must be destroyed.

Example 13-3 shows how to use a drag and drop text cursor with a drag and drop object.

Example 13-3. Using drag and drop text cursors

```
if (drag and drop started) {
  type = event_ctrl_is_down(event) ? CURSOR_DUPLICATE : CURSOR_MOVE;
  neutral_drag_cursor = xv_create(window, CURSOR,
           CURSOR_STRING, selected_string,
           CURSOR_DRAG_TYPE, type,
           CURSOR_DRAG_STATE, CURSOR_NEUTRAL,
           NULL);
  accept_drag_cursor = xv_create(window, CURSOR,
           CURSOR_STRING, selected_string,
           CURSOR_DRAG_TYPE, type,
           CURSOR_DRAG_STATE, CURSOR_ACCEPT,
           NULL);
  xv_set(dnd_object,
           DND_CURSOR, neutral_drag_cursor,
           DND_ACCEPT_CURSOR, accept_cursor,
           NULL);
}
```

*Note: The current drag and drop protocol does not support a reject cursor.

13.6 Cursor Package Summary

Table 13-1 shows the CURSOR package procedure. Table 13-2 lists the attributes for the CURSOR package. This information is described fully in the *XView Reference Manual*.

Table 13-1. Cursor Procedure

cursor_copy()

Table 13-2. Cursor Attributes

CURSOR_BACKGROUND_COLOR	CURSOR_OP
CURSOR_DRAG_STATE	CURSOR_SRC_CHAR
CURSOR_DRAG_TYPE	CURSOR_STRING
CURSOR_FOREGROUND_COLOR	CURSOR_XHOT
CURSOR_IMAGE	CURSOR_YHOT
CURSOR_MASK_CHAR	
XV_SHOW	XV_XID