# F
# Example Programs

Example Programs

This appendix contains nine example programs that supplement the programs in the chapters:

- *item_move.c*
- *scroll_cells2.c*
- *menu_dir2.c*
- *type_font.c*
- *fonts.c*
- *x_draw.c*
- *Logo.c*
- *Bitmap.c*
- *panel_dnd.c*

Some of these programs are extensions to programs presented earlier in this book; they are listed here to demonstrate extended usage. Other programs in this appendix attempt to integrate features from unrelated XView packages that exceeded the scope of a particular chapter.

## F.1 item_move.c

The first program demonstrates how you can use an event handler within a panel to allow the user to create panel items, move them around within the panel, and delete them. Chapter 7, *Panels*, discusses specifics about how panels work. Chapter 5, *Canvases and Openwin*, discusses the canvas and openwin issues, since the panel is subclassed from those packages.

*Example F-1. The item_move.c program*

```
/*
 * item_move.c
 *    Move items around in a panel using an interpose event handler
 * specific to the panel.  Two panels are created -- the left panel
 * contains panel buttons that allow you to create certain types of
 * items that are put in the second panel.  Use the MENU (right)
```

*Example F-1. The item_move.c program (continued)*

```
 * mouse button to move items around in the second panel.
 */
#include <stdio.h>
#include <xview/xview.h>
#include <xview/panel.h>

/* We need handles to the base frame and a panel -- instead of
 * using global variables, we're going to attach the objects to
 * the objects which need to reference them.  Attach using
 * XV_KEY_DATA -- here are the keys.
 */
#define PACKAGE_KEY     100
#define FRAME_KEY       101
#define PANEL_KEY       102

main(argc, argv)
int argc;
char *argv[];
{
    Frame        frame;
    Panel        panel;
    Xv_Window          window;
    Panel_item   create_text, item;
    Notify_value my_event_proc();
    int          create_item();
    char         buf[64];

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

    sprintf(buf, "%s:  Use MENU (Right) Button To Move Items", argv[0]);
    frame = (Frame)xv_create(XV_NULL, FRAME,
        FRAME_LABEL,            buf,
        FRAME_SHOW_FOOTER,      TRUE,
        NULL);
    /*
     * Create panel for known panel items.  Layout panel vertically.
     */
    panel = (Panel)xv_create(frame, PANEL,
        PANEL_LAYOUT,           PANEL_VERTICAL,
        NULL);
    /*
     * Create text panel item, attach the frame as client data for
     * use by the notify procedure create_item().  Text items inherit
     * the layout of "label" and "value" from its parent panel.
     * override for the text item by setting PANEL_LAYOUT explicitly.
     */
    create_text = (Panel_item)xv_create(panel, PANEL_TEXT,
      XV_X,            0,
      XV_Y,            20,
        PANEL_LABEL_STRING,     "Create Button:",
        PANEL_NOTIFY_PROC,      create_item,
        PANEL_LAYOUT,           PANEL_HORIZONTAL,
        PANEL_VALUE_DISPLAY_LENGTH,     10,
        NULL);
    /*
     * Create panel button to determine which type of button to create --
```

*Example F-1. The item_move.c program (continued)*

```
     * a button, message, or text item.  See create_item().
     */
    item = (Panel_item)xv_create(panel, PANEL_CHOICE,
      XV_X,            0,
      XV_Y,            50,
        PANEL_DISPLAY_LEVEL,    PANEL_CURRENT,
        PANEL_LAYOUT,           PANEL_HORIZONTAL,
        PANEL_LABEL_STRING,     "Item type",
        PANEL_CHOICE_STRINGS,   "Button", "Message", "Text", NULL,
        NULL);
    window_fit(panel);

    /* Create a new panel to be used for panel creation.  The panel
     * from above is no longer referenced.  The panel created here
     * is the panel used throughout the rest of this program.  To
     * add confusion, "panel" is used as the handle of this panel, too.
     * The panel referenced in WIN_RIGHT_OF and XV_HEIGHT is the old
     * one since the new one hasn't been created yet.
     */
    panel = (Panel)xv_create(frame, PANEL,
        WIN_RIGHT_OF,           canvas_paint_window(panel),
        XV_WIDTH,               300,
        XV_HEIGHT,              xv_get(panel, XV_HEIGHT),
        WIN_BORDER,          TRUE,
        XV_KEY_DATA,            PANEL_KEY,      panel,
        NULL);

    /* Install event handling routine for the panel.  This must be done
     *      by an interpose function to make sure that the ACTION_MENU
     *      event is not consumed by the first panel before it has a chance
     *      to get to the second panel's event proc:  my_event_proc.
     */
    notify_interpose_event_func(panel, my_event_proc, NOTIFY_SAFE);

    /* attach various items to the text item for text_select() */
    xv_set(create_text,
        XV_KEY_DATA,            FRAME_KEY,      frame,
        XV_KEY_DATA,            PACKAGE_KEY,    item,
        XV_KEY_DATA,            PANEL_KEY,      panel,
        NULL);
    window_fit(frame);
    xv_main_loop(frame);
}

/*
 * Process events for panel's subwindow.  This routine gets -all-
 * events that occur in the panel subwindow but passes them on to
 * the normal event dispatcher when the interposed function has been
 * completed.  The notify function, my_event_proc, is only
 * interested in MENU button events that happen on top of panel items.
 * When the user clicks and _drags_ the MENU button on a panel item,
 * the item is moved to where the mouse moves to.
 */
Notify_value
my_event_proc(panel, event, arg, type)
Panel                   panel;
```

*Example F-1. The item_move.c program (continued)*

```
Event            *event;
Notify_arg        arg;
Notify_event_type type;
{
    static Panel_item   item;
    static int  x_offset, y_offset;
    Frame       frame = (Frame)xv_get(panel, XV_OWNER);
    Rect        *rect, *item_rect;
    char        buf[64];

    /*
     * If the mouse is dragging an item, reset its new location.
     */
    if (event_action(event) == LOC_DRAG && item) {
        Panel_item pi;
        Rect r;
        /*
         * Get the rect of item, then *copy* it -- never change data
         * returned by xv_get().  Modify the copied rect reflecting
         * new X,Y position of panel item and check to see if it
         * intersects with any existing panel items.
         */
        rect = (Rect *)xv_get(item, XV_RECT);
        rect_construct(&r, /* see <xview/rect.h> for macros */
            rect->r_left, rect->r_top, rect->r_width, rect->r_height);
        r.r_left = event->ie_locx - x_offset;
        r.r_top = event->ie_locy - y_offset;
        PANEL_EACH_ITEM(panel, pi)
            if (pi == item)
                continue;
            /* don't let panel items overlap */
            item_rect = (Rect *)xv_get(pi, XV_RECT);
            if (rect_intersectsrect(item_rect, &r))
                return;
        PANEL_END_EACH
        /* no overlap -- move panel item. */
        xv_set(item,
            PANEL_ITEM_X, r.r_left,
            PANEL_ITEM_Y, r.r_top,
            NULL);
    }

    /* If it's not the MENU button, we're not interested,
     * so allow the event to be passed on to the notifier
     * for normal event handling.
     */
    if (event_action(event) != ACTION_MENU) {
      notify_next_event_func(panel, (Notify_event) event, arg, type);
      return;
    }

    /*
     * next two cases is MENU button just-down or just-released
     */
    if (event_is_down(event)) {
        /* Right (MENU) button down on an item -- determine panel item */
```

*Example F-1. The item_move.c program (continued)*

```
      if ( (xv_get((panel), PANEL_FIRST_ITEM) ) == NULL ) {
              sprintf(buf, "No panel items are currently in the panel.");
           xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
      }
        PANEL_EACH_ITEM(panel, item)
            rect = (Rect *)xv_get(item, XV_RECT);
            if (rect_includespoint(rect,
                event->ie_locx, event->ie_locy)) {
                x_offset = event->ie_locx - rect->r_left;
                y_offset = event->ie_locy - rect->r_top;
              sprintf(buf, "Panel item found.");
           xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
                break;
            }
          else {
              sprintf(buf, "The cursor is not over any panel item.");
           xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
            }
        PANEL_END_EACH
        if (item)
            sprintf(buf, "Moving item: '%s'",
                (char *)xv_get(item, PANEL_LABEL_STRING));
        else
            buf[0] = 0;
    } else if (item) {
        char *name = (char *)xv_get(item, PANEL_LABEL_STRING);

        /* test if item is inside panel by comparing XV_RECTs */
        rect = (Rect *)xv_get(panel, XV_RECT);
        if (!rect_includespoint(rect,
            event->ie_locx + rect->r_left,
            event->ie_locy + rect->r_top)) {
            /* item is outside the panel -- remove item */
            xv_destroy(item);
            sprintf(buf, "Removed '%s' from panel", name);
        } else
            sprintf(buf, "'%s' moved to %d %d", name,
                (int)xv_get(item, XV_X), (int)xv_get(item, XV_Y));
        /* set "item" to null so that new drag
         * events don't attempt to move old item.
         */
        item = NULL;
    }
    xv_set(frame, FRAME_LEFT_FOOTER, buf, NULL);
}

/*
 * Callback routine for all panel buttons.
 * If the panel item is the text item, determine the name of the new
 * panel button the user wishes to create.  Loop through all the
 * existing panel items looking for one with the same label.  If so,
 * return PANEL_NONE and set the frame's footer with an error message.
 * Otherwise, create a new panel item with the label, reset the text
 * item value and return PANEL_NEXT.
 */
int
```

*Example F-1. The item_move.c program (continued)*

```
create_item(item, event)
Panel_item item;
Event *event;
{
    Xv_pkg      *pkg;
    Panel       panel = (Panel)xv_get(item, XV_KEY_DATA, PANEL_KEY);
    Frame       frame = (Frame)xv_get(item, XV_KEY_DATA, FRAME_KEY);
    Panel_item  pi, pkg_item;
    char        buf[64];
    int         selected();

    pkg_item = (Panel_item)xv_get(item, XV_KEY_DATA, PACKAGE_KEY);
    (void) strncpy(buf, (char *)xv_get(item, PANEL_VALUE), sizeof buf);
    if (!buf[0])
        return PANEL_NONE;
    switch((int)xv_get(pkg_item, PANEL_VALUE)) {
        case 1: pkg = PANEL_MESSAGE; break;
        case 2: pkg = PANEL_TEXT; break;
        default: pkg = PANEL_BUTTON;
    }
    /* loop thru all panel items and check for item with same name */
    PANEL_EACH_ITEM(panel, pi)
        if (!strcmp(buf, (char *)xv_get(pi, PANEL_LABEL_STRING))) {
            xv_set(frame, FRAME_LEFT_FOOTER, "Label Taken", NULL);
            return PANEL_NONE;
        }
    PANEL_END_EACH
    (void) xv_create(panel, pkg,
        PANEL_LABEL_STRING,     buf,
        PANEL_NOTIFY_PROC,      selected,
        XV_KEY_DATA,            FRAME_KEY,      frame,
        /* only for text items, but doesn't affect other items */
        PANEL_VALUE_DISPLAY_LENGTH, 10,
        PANEL_LAYOUT,           PANEL_HORIZONTAL,
        NULL);
    xv_set(item, PANEL_VALUE, "", NULL);
    return PANEL_NEXT;
}

/*
 * For panel buttons. return XV_OK or XV_ERROR if the item was
 * selected using the left mouse button or not.
 */
int
selected(item, event)
Panel_item item;
Event *event;
{
    Frame       frame = (Frame)xv_get(item, XV_KEY_DATA, FRAME_KEY);
    char        buf[64];

    if (event_action(event) == ACTION_SELECT) {
        sprintf(buf, "'%s' selected", xv_get(item, PANEL_LABEL_STRING));
        xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
        return XV_OK;
    }
```

*Example F-1. The item_move.c program (continued)*

```
    return XV_ERROR;
}
```

## F.2 scroll_cells2.c

This program is based heavily on *scroll_cells.c*, which is found in Chapter 10, *Scrollbars*. This version of the program deals with resize events. When a resize occurs, the `object`, `view`, and `page` length attributes are set to correctly reflect the size of the scrollbar with respect to the object it scrolls.

Example Programs

*Example F-2. The scroll_cells2.c program*

```
/*
 * scroll_cells2.c -- scroll a bitmap of cells around in a canvas.
 * This is a simplified version of scroll_cells.c graphically. That
 * is, it does not display icons, just rows and columns of cells.
 * The difference with this version is that it attempts to accommodate
 * resize events not addressed in the scroll_cells.c.
 * This new function is at the end of the file.
 */
#include <stdio.h>
#include <X11/X.h>
#include <X11/Xlib.h>   /* Using Xlib graphics */
#include <xview/xview.h>
#include <xview/canvas.h>
#include <xview/scrollbar.h>
#include <xview/xv_xrect.h>

#define CELL_WIDTH              64
#define CELL_HEIGHT             64
#define CELLS_PER_HOR_PAGE      5 /* when paging w/scrollbar */
#define CELLS_PER_VER_PAGE      5 /* when paging w/scrollbar */
#define CELLS_PER_ROW           16
#define CELLS_PER_COL           16

Pixmap          cell_map;       /* pixmap copied onto canvas window */
Scrollbar       horiz_scrollbar;
Scrollbar       vert_scrollbar;
GC              gc;             /* General usage GC */

main(argc, argv)
int argc;
char *argv[ ];
{
    Frame       frame;
    Canvas      canvas;
    void        repaint_proc(), resize_proc();

    /* Initialize, create frame and canvas... */
    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);
```

*Example F-2. The scroll_cells2.c program (continued)*

```
    frame = xv_create(XV_NULL, FRAME,
        FRAME_LABEL,            argv[0],
        FRAME_SHOW_FOOTER,      TRUE,
        NULL);

    canvas = xv_create(frame, CANVAS,
        /* make subwindow the size of a "page" */
        XV_WIDTH,               CELL_WIDTH * CELLS_PER_HOR_PAGE,
        XV_HEIGHT,              CELL_HEIGHT * CELLS_PER_VER_PAGE,
        /* canvas is same size as window */
        CANVAS_WIDTH,           CELL_WIDTH * CELLS_PER_HOR_PAGE,
        CANVAS_HEIGHT,          CELL_HEIGHT * CELLS_PER_VER_PAGE,
        /* don't retain window -- we'll repaint it all the time */
        CANVAS_RETAINED,        FALSE,
        /* We're using Xlib graphics calls in repaint_proc() */
        CANVAS_X_PAINT_WINDOW,  TRUE,
        CANVAS_REPAINT_PROC,    repaint_proc,
        CANVAS_RESIZE_PROC,     resize_proc,
        OPENWIN_AUTO_CLEAR,     FALSE,
        NULL);

    /*
     * Create scrollbars attached to the canvas. When user clicks
     * on cable, page by the page size (PAGE_LENGTH). Scrolling
     * should move cell by cell, not by one pixel (PIXELS_PER_UNIT).
     */
    vert_scrollbar = xv_create(canvas, SCROLLBAR,
        SCROLLBAR_DIRECTION,            SCROLLBAR_VERTICAL,
        SCROLLBAR_PIXELS_PER_UNIT,      CELL_HEIGHT,
        NULL);
    horiz_scrollbar = xv_create(canvas, SCROLLBAR,
        SCROLLBAR_DIRECTION,            SCROLLBAR_HORIZONTAL,
        SCROLLBAR_PIXELS_PER_UNIT,      CELL_WIDTH,
        NULL);

    /*
     * create pixmap and draw cells into it.  This portion of the
     * program could use XCopyArea to render real bitmaps whose sizes
     * do not exceed whatever CELL_WIDTH and CELL_HEIGHT are defined
     * to be.  The cell_map will be copied into the window via
     * XCopyPlane in the repaint procedure.
     */
    {
        short           x, y, pt = 0;
        XPoint          points[256];
        XGCValues       gcvalues;
        Display *dpy = (Display *)xv_get(canvas, XV_DISPLAY);

        cell_map = XCreatePixmap(dpy, DefaultRootWindow(dpy),
            CELLS_PER_ROW * CELL_WIDTH + 1,
            CELLS_PER_COL * CELL_HEIGHT + 1,
            1); /* We only need a 1-bit deep pixmap */

        /* Create the gc for the cell_map -- since it is 1-bit deep,
         * use 0 and 1 for fg/bg values.  Also, limit number of
         * events generated by setting graphics exposures to False.
```

*Example F-2. The scroll_cells2.c program (continued)*

```
         */
        gcvalues.graphics_exposures = False;
        gcvalues.background = 0;
        gcvalues.foreground = 1;
        gc = XCreateGC(dpy, cell_map,
            GCForeground|GCBackground|GCGraphicsExposures, &gcvalues);

        /* dot every other pixel */
        for (x = 0; x <= CELL_WIDTH * CELLS_PER_ROW; x += 2)
            for (y = 0; y <= CELL_HEIGHT * CELLS_PER_COL; y += 2) {
                if (x % CELL_WIDTH != 0 && y % CELL_HEIGHT != 0)
                    continue;
                points[pt].x = x, points[pt].y = y;
                if (++pt == sizeof points / sizeof points[0]) {
                    XDrawPoints(dpy, cell_map, gc,
                        points, pt, CoordModeOrigin);
                    pt = 0;
                }
            }
        if (pt != sizeof points) /* flush out the remaining points */
            XDrawPoints(dpy, cell_map, gc,
                        points, pt, CoordModeOrigin);
        /* label each cell indicating the its coordinates */
        for (x = 0; x < CELLS_PER_ROW; x++)
            for (y = 0; y < CELLS_PER_COL; y++) {
                char buf[8];
                sprintf(buf, "%d,%d", x+1, y+1);
                XDrawString(dpy, cell_map, gc,
                    x * CELL_WIDTH + 5, y * CELL_HEIGHT + 25,
                    buf, strlen(buf));
            }
        /* we're now done with the cell_map, so free gc and
         * create a new one based on the window that will use it.
         */
        XFreeGC(dpy, gc);
        gcvalues.background = WhitePixel(dpy, DefaultScreen(dpy));
        gcvalues.foreground = BlackPixel(dpy, DefaultScreen(dpy));
        gcvalues.plane_mask = 1L;
        gc = XCreateGC(dpy, DefaultRootWindow(dpy),
            GCForeground|GCBackground|GCGraphicsExposures, &gcvalues);
    }

    /* shrink frame to minimal size and start notifier */
    window_fit(frame);
    xv_main_loop(frame);
}

/*
 * The repaint procedure is called whenever repainting is needed in
 * a paint window.  Since the canvas is not retained, this routine
 * is going to be called any time the user scrolls the canvas.  The
 * canvas will handle repainting the portion of the canvas that
 * was in view and has scrolled onto another viewable portion of
 * the window.  The xrects parameter will cover the new areas that
 * were not in view before and have just scrolled into view.  If
 * the window resizes or if the window is exposed by other windows
```

Example Programs

*Example F-2. The scroll_cells2.c program (continued)*

```
 * disappearing or cycling thru the window tree, then the number
 * of xrects will be more than one and we'll have to copy the new
 * areas one by one.  Clipping isn't necessary since the areas to
 * be rendered are set by the xrects value.
 */
void
repaint_proc(canvas, paint_window, dpy, win, xrects)
Canvas          canvas;
Xv_Window       paint_window;
Display         *dpy;
Window          win;
Xv_xrectlist    *xrects;
{
    int x, y;

    x = (int)xv_get(horiz_scrollbar, SCROLLBAR_VIEW_START);
    y = (int)xv_get(vert_scrollbar, SCROLLBAR_VIEW_START);

    for (xrects->count--; xrects->count >= 0; xrects->count--) {
        printf("top-left cell = %d, %d -- %d,%d %d,%d0, x+1, y+1,
            xrects->rect_array[xrects->count].x,
            xrects->rect_array[xrects->count].y,
            xrects->rect_array[xrects->count].width,
            xrects->rect_array[xrects->count].height);

        XCopyPlane(dpy, cell_map, win, gc,
            x * CELL_WIDTH,
            y * CELL_HEIGHT,
            xv_get(paint_window, XV_WIDTH),
            xv_get(paint_window, XV_HEIGHT),
            0, 0, 1L);
    }
}
/*
 * If the application is resized, then we may wish to reset the
 * paging and viewing parameters for the scrollbars.
 */
void
resize_proc(canvas, new_width, new_height)
Canvas canvas;
int new_width, new_height;
{
    int page_w = (int)(new_width/CELL_WIDTH);
    int page_h = (int)(new_height/CELL_HEIGHT);

    if (!vert_scrollbar || !horiz_scrollbar)
        return;

    printf("new width/height in cells: w = %d, h = %d0,
        page_w, page_h);

    xv_set(horiz_scrollbar,
        SCROLLBAR_OBJECT_LENGTH,        CELLS_PER_ROW,
        SCROLLBAR_PAGE_LENGTH,          page_w,
        SCROLLBAR_VIEW_LENGTH,          page_w,
        NULL);
```

*Example F-2. The scroll_cells2.c program (continued)*

```
    xv_set(vert_scrollbar,
        SCROLLBAR_OBJECT_LENGTH,        CELLS_PER_COL,
        SCROLLBAR_PAGE_LENGTH,          page_h,
        SCROLLBAR_VIEW_LENGTH,          page_h,
        NULL);
}
```

## F.3 menu_dir2.c



In Chapter 11, *Menus*, the program *menu_dir.c* demonstrates the use of an XView menu in a canvas subwindow. A menu is brought up with the MENU mouse button and displays menu choices representing the files in the directory. The problem with *menu_dir.c* is that the entire menu cascade is created for all the subdirectories at the very beginning of the program. If the directory stack is very deep, it could take a very long time to build. You could also run out of memory in the process. Further, if the contents of the directory tree is dynamic, the menu entries could become invalid over time.

These problems are solved in *menu_dir2.c* because it creates only the top-level menu. For each directory entry under the top-level, rather than creating an associated pullright menu, a `MENU_GEN_PULLRIGHT` procedure is specified. This routine creates that menu only at the time it is needed. So, when the user invokes the menu and tries to descend into a submenu representing a subdirectory in the directory tree, only then is the directory entry searched and a new submenu created. When the user backs out of the menu, the menu is destroyed—attempting to re-enter the submenu causes the process to be repeated.

An exercise for the ambitious programmer would be to modify this program so that the submenus are not destroyed until the entire menu cascade has been dismissed. This enhancement would optimize the perceived performance of the program for the user because, once a directory subpath has been searched and a menu created, the menu is cached so that re-entry into the same submenu would be instantaneous.

*Example F-3. The menu_dir2.c program*

```
/*
 * menu_dir2.c -
 * Demonstrate the use of an XView menu in a canvas subwindow.
 * A menu is brought up with the MENU mouse button and displays
 * menu choices representing the files in the directory.  If a
 * directory entry is found, a new pullright item is created with
 * that subdir as the pullright menu's contents.  This implementation
 * creates directories on an as-needed basis.  Thus, we provide a
 * MENU_GEN_PULLRIGHT procedure.
 *
 * argv[1] indicates which directory to start from.
 */
#include <xview/xview.h>
#include <xview/canvas.h>
#include <sys/stat.h>
#include <sys/dir.h>
```

*Example F-3. The menu_dir2.c program (continued)*

```
#include <X11/Xos.h>
#ifndef MAXPATHLEN
#include <sys/param.h>
#endif /* MAXPATHLEN */

Frame   frame;

main(argc,argv)
int     argc;
char    *argv[ ];
{
    Canvas      canvas;
    extern void exit();
    void        my_event_proc();
    Menu        menu;
    Menu_item   mi, add_path_to_menu();

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

    frame = (Frame)xv_create(NULL, FRAME,
        FRAME_LABEL,            argv[1]? argv[1] : "cwd",
        FRAME_SHOW_FOOTER,      TRUE,
        NULL);
    canvas = (Canvas)xv_create(frame, CANVAS,
        FRAME_LABEL,    argv[0],
        XV_WIDTH,       400,
        XV_HEIGHT,      100,
        NULL);

    mi = add_path_to_menu(argc > 1? argv[1] : ".");
    menu = (Menu)xv_get(mi, MENU_PULLRIGHT);
    /* We no longer need the item since we have the menu from it */
    xv_destroy(mi);

    /* associate the menu to the canvas win for easy etreival */
    xv_set(canvas_paint_window(canvas),
        WIN_CONSUME_EVENTS,     WIN_MOUSE_BUTTONS, NULL,
        WIN_EVENT_PROC,         my_event_proc,
        WIN_CLIENT_DATA,        menu,
        NULL);

    window_fit(frame);
    window_main_loop(frame);
}

/*
 * my_action_proc - display the selected item in the frame footer.
 */
void
my_action_proc(menu, menu_item)
Menu    menu;
Menu_item       menu_item;
{
    xv_set(frame,
        FRAME_LEFT_FOOTER,      xv_get(menu_item, MENU_STRING),
        NULL);
```

*Example F-3. The menu_dir2.c program (continued)*

```
}

/*
 * Call menu_show() to display menu on right mouse button push.
 */
void
my_event_proc(canvas, event)
Canvas  canvas;
Event *event;
{
    if ((event_id(event) == MS_RIGHT) && event_is_down(event)) {
        Menu menu = (Menu)xv_get(canvas, WIN_CLIENT_DATA);
        menu_show(menu, canvas, event, NULL);
    }
}

/*
 * return an allocated char * that points to the last item in a path.
 */
char *
getfilename(path)
char *path;
{
    char *p;

    if (p = rindex(path, '/'))
        p++;
    else
        p = path;
    return strcpy(malloc(strlen(p)+1), p);
}

/* gen_pullright() is called in the following order:
 *   Pullright menu needs to be displayed. (MENU_PULLRIGHT)
 *   Menu is about to be dismissed (MENU_DISPLAY_DONE)
 *      User made a selection (before menu notify function)
 *      After the notify routine has been called.
 * The above order is done whether or not the user makes a
 * menu selection.
 */
Menu
gen_pullright(mi, op)
Menu_item mi;
Menu_generate op;
{
    Menu menu;
    Menu_item new, old = mi;
    char buf[MAXPATHLEN];

    if (op == MENU_DISPLAY) {
        menu = (Menu)xv_get(mi, MENU_PARENT);
        sprintf(buf, "%s/%s",
            xv_get(menu, MENU_CLIENT_DATA), xv_get(mi, MENU_STRING));
        /* get old menu and free it -- we're going to build another */
        if (menu = (Menu)xv_get(mi, MENU_PULLRIGHT)) {
            free(xv_get(menu, MENU_CLIENT_DATA));
```

Example Programs

*Example F-3. The menu_dir2.c program (continued)*

```
            xv_destroy(menu);
        }
        if (new = add_path_to_menu(buf)) {
            menu = (Menu)xv_get(new, MENU_PULLRIGHT);
            xv_destroy(new);
            return menu;
        }
    }
    if (!(menu = (Menu)xv_get(mi, MENU_PULLRIGHT)))
            menu = (Menu)xv_create(NULL, MENU,
                MENU_STRINGS, "Couldn't build a menu.", NULL,
                NULL);
    return menu;
}

/*
 * The path passed in is scanned via readdir().  For each file in the
 * path, a menu item is created and inserted into a new menu.  That
 * new menu is made the PULLRIGHT_MENU of a newly created panel item
 * for the path item originally passed it.  Since this routine is
 * recursive, a new menu is created for each subdirectory under the
 * original path.
 */
Menu_item
add_path_to_menu(path)
char *path;
{
    DIR                 *dirp;
    struct direct       *dp;
    struct stat         s_buf;
    Menu_item           mi;
    Menu                next_menu;
    char                buf[MAXPATHLEN];
    static int          recursion;

    /* don't add a folder to the list if user can't read it */
    if (stat(path, &s_buf) == -1 || !(s_buf.st_mode & S_IREAD))
        return NULL;
    if (s_buf.st_mode & S_IFDIR) {
        int cnt = 0;
        if (!(dirp = opendir(path)))
            /* don't bother adding to list if we can't scan it */
            return NULL;
        if (recursion)
            return (Menu_item)-1;
        recursion++;
        next_menu = (Menu)xv_create(XV_NULL, MENU, NULL);
        while (dp = readdir(dirp))
            if (strcmp(dp->d_name, ".") && strcmp(dp->d_name, "..")) {
                (void) sprintf(buf, "%s/%s", path, dp->d_name);
                mi = add_path_to_menu(buf);
                if (!mi || mi == (Menu_item)-1) {
                    int do_gen_pullright = (mi == (Menu_item)-1);
                    /* unreadable file or dir - deactivate item */
                    mi = (Menu_item)xv_create(XV_NULL, MENUITEM,
                        MENU_STRING,  getfilename(dp->d_name),
```

Example F-3. The menu_dir2.c program (continued)

```
                        MENU_RELEASE,
                        MENU_RELEASE_IMAGE,
                        NULL);
                    if (do_gen_pullright)
                        xv_set(mi,
                            MENU_GEN_PULLRIGHT, gen_pullright,
                            NULL);
                    else
                        xv_set(mi, MENU_INACTIVE, TRUE, NULL);
                }
                xv_set(next_menu, MENU_APPEND_ITEM, mi, NULL);
                cnt++;
            }
        closedir(dirp);
        mi = (Menu_item)xv_create(XV_NULL, MENUITEM,
            MENU_STRING,        getfilename(path),
            MENU_RELEASE,
            MENU_RELEASE_IMAGE,
            MENU_NOTIFY_PROC,   my_action_proc,
            NULL);
        if (!cnt) {
            xv_destroy(next_menu);
            /* An empty or unsearchable directory - deactivate item */
            xv_set(mi, MENU_INACTIVE, TRUE, NULL);
        } else {
            xv_set(next_menu,
                MENU_TITLE_ITEM, strcpy(malloc(strlen(path)+1), path),
                MENU_CLIENT_DATA, strcpy(malloc(strlen(path)+1), path),
                NULL);
            xv_set(mi, MENU_PULLRIGHT, next_menu, NULL);
        }
        recursion--;
        return mi;
    }
    return (Menu_item)xv_create(NULL, MENUITEM,
        MENU_STRING,            getfilename(path),
        MENU_RELEASE,
        MENU_RELEASE_IMAGE,
        MENU_NOTIFY_PROC,       my_action_proc,
        NULL);
}
```

## F.4 type_font.c

This very simple program captures keyboard events in a canvas and uses `XDrawString()` to render what the user types. It also looks for backspacing. This is not intended to replace the text subwindow in any way, but rather to demonstrate how some text rendering functions can be implemented. See Chapter 16, *Fonts*, for more information about fonts.

*Example F-4. The type_font.c program*

```
/*
 * simple_font.c -- very simple program showing how to render text
 * using fonts loaded by XView.
 */
#include <ctype.h>
#include <X11/X.h>
#include <X11/Xlib.h>
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/font.h>

Display *dpy;
GC      gc;
XFontStruct *font_info;

main(argc, argv)
int     argc;
char    *argv[ ];
{
    Frame       frame;
    Panel       panel;
    Canvas      canvas;
    XGCValues   gcvalues;
    Xv_Font     font;
    void        my_event_proc();
    extern void exit();

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

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

    panel = (Panel)xv_create(frame, PANEL,
        PANEL_LAYOUT,           PANEL_VERTICAL,
        NULL);
    xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Quit",
        PANEL_NOTIFY_PROC,      exit,
        NULL);
    window_fit(panel);

    canvas = (Canvas)xv_create(frame, CANVAS,
        XV_WIDTH,               400,
        XV_HEIGHT,              200,
        CANVAS_X_PAINT_WINDOW,  TRUE,
        NULL);
    xv_set(canvas_paint_window(canvas),
        WIN_EVENT_PROC,         my_event_proc,
        NULL);

    window_fit(frame);

    dpy = (Display *)xv_get(frame, XV_DISPLAY);
    font = (Xv_Font)xv_get(frame, XV_FONT);
    font_info = (XFontStruct *)xv_get(font, FONT_INFO);
```

*Example F-4. The type_font.c program (continued)*

```
    gcvalues.font = (Font)xv_get(font, XV_XID);
    gcvalues.foreground = BlackPixel(dpy, DefaultScreen(dpy));
    gcvalues.background = WhitePixel(dpy, DefaultScreen(dpy));
    gcvalues.graphics_exposures = False;
    gc = XCreateGC(dpy, RootWindow(dpy, DefaultScreen(dpy)),
        GCForeground | GCBackground | GCFont | GCGraphicsExposures, &gcvalues);

    xv_main_loop(frame);
}

void
my_event_proc(win, event)
Xv_Window win;
Event *event;
{
    static int x = 10, y = 10;
    Window xwin = (Window)xv_get(win, XV_XID);
    char c;

    if (event_is_up(event))
        return;

    if (event_is_ascii(event)) {
        c = (char)event_id(event);
        if (c == ’0 || c == ’) {
            y += font_info->max_bounds.ascent +
                        font_info->max_bounds.descent;
            x = 10;
        } else if (c == 7 || c == 127) { /* backspace or delete */
            if (x > 10)
                x -= XTextWidth(font_info, "m", 1);
            /* use XDrawImageString to overwrite previous text */
            XDrawImageString(dpy, xwin, gc, x, y, "  ", 2);
        } else {
            XDrawString(dpy, xwin, gc, x, y, &c, 1);
            x += XTextWidth(font_info, &c, 1);
        }
    } else if (event_action(event) == ACTION_SELECT) {
        x = event_x(event);
        y = event_y(event);
    }
}
```

## F.5 fonts.c

This program is similar to *type_font.c* above. However, *fonts.c* provides an interface for the user to pick and choose from a subset of the font families and styles available on the X server. If a font "name" is specified, then the family, style and size choices are ignored. Using the SELECT button on the canvas window positions the current typing location at the *x*,*y* coordinates of the button-down event. The characters typed are printed in the current font.

*Example F-5. The fonts.c program*

```
/*
 * fonts.c -- provide an interface for the user to pick and choose
 * between font families and styles known to XView.  The program
 * provides several panel buttons to choose between font types, and
 * a canvas window in which the user can type.  The characters typed
 * are printed in the current font.  If a font "name" is specified,
 * then the family, style and size are ignored.  Using the SELECT
 * button on the canvas window positions the current typing location
 * at the x,y coordinates of the button-down event.
 */
#include <ctype.h>
#include <X11/X.h>
#include <X11/Xlib.h>
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/font.h>

Display *dpy;
GC      gc;
XFontStruct *cur_font;
Panel_item family_item, style_item, scale_item, name_item;
int canvas_width;

main(argc, argv)
int     argc;
char    *argv[];
{
    Frame       frame;
    Panel       panel;
    Canvas      canvas;
    XGCValues   gcvalues;
    Xv_Font     font;
    void        change_font();
    void        my_event_proc(), my_resize_proc();
    int         change_font_by_name();
    extern void exit();

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

    frame = (Frame)xv_create(XV_NULL, FRAME,
        FRAME_LABEL,            argv[0],
        FRAME_SHOW_FOOTER,      TRUE,
        NULL);
```

*Example F-5. The fonts.c program (continued)*

```
panel = (Panel)xv_create(frame, PANEL,
    PANEL_LAYOUT,           PANEL_VERTICAL,
    NULL);
(void) xv_create(panel, PANEL_BUTTON,
    PANEL_LABEL_STRING,     "Quit",
    PANEL_NOTIFY_PROC,      exit,
    NULL);
family_item = (Panel_item)xv_create(panel, PANEL_CHOICE,
    PANEL_LABEL_STRING,     "Family",
    PANEL_LAYOUT,           PANEL_HORIZONTAL,
    PANEL_DISPLAY_LEVEL,    PANEL_CURRENT,
    PANEL_CHOICE_STRINGS,
        FONT_FAMILY_DEFAULT, FONT_FAMILY_DEFAULT_FIXEDWIDTH,
        FONT_FAMILY_LUCIDA, FONT_FAMILY_LUCIDA_FIXEDWIDTH,
        FONT_FAMILY_ROMAN, FONT_FAMILY_SERIF, FONT_FAMILY_COUR,
        FONT_FAMILY_CMR, FONT_FAMILY_GALLENT,
        FONT_FAMILY_OLGLYPH, FONT_FAMILY_OLCURSOR, NULL,
    PANEL_NOTIFY_PROC,      change_font,
    NULL);
style_item = (Panel_item)xv_create(panel, PANEL_CHOICE,
    PANEL_LABEL_STRING,     "Style",
    PANEL_LAYOUT,           PANEL_HORIZONTAL,
    PANEL_DISPLAY_LEVEL,    PANEL_CURRENT,
    PANEL_CHOICE_STRINGS,
        FONT_STYLE_DEFAULT, FONT_STYLE_NORMAL, FONT_STYLE_BOLD,
        FONT_STYLE_ITALIC, FONT_STYLE_BOLD_ITALIC, NULL,
    PANEL_NOTIFY_PROC,      change_font,
    NULL);
scale_item = (Panel_item)xv_create(panel, PANEL_CHOICE,
    PANEL_LABEL_STRING,     "Scale",
    PANEL_LAYOUT,           PANEL_HORIZONTAL,
    PANEL_DISPLAY_LEVEL,    PANEL_CURRENT,
    PANEL_CHOICE_STRINGS,
        "Small", "Medium", "Large", "X-Large", NULL,
    PANEL_NOTIFY_PROC,      change_font,
    NULL);
name_item = (Panel_item)xv_create(panel, PANEL_TEXT,
    PANEL_LABEL_STRING,     "Font Name:",
    PANEL_LAYOUT,           PANEL_HORIZONTAL,
    PANEL_VALUE_DISPLAY_LENGTH, 20,
    PANEL_NOTIFY_PROC,      change_font_by_name,
    NULL);
window_fit(panel);

canvas = (Canvas)xv_create(frame, CANVAS,
    XV_WIDTH,               400,
    XV_HEIGHT,              200,
    CANVAS_X_PAINT_WINDOW,  TRUE,
    CANVAS_RESIZE_PROC,     my_resize_proc,
    NULL);
xv_set(canvas_paint_window(canvas),
    WIN_EVENT_PROC,         my_event_proc,
    WIN_CONSUME_EVENT,      LOC_WINENTER,
    NULL);

window_fit(frame);
```

Example Programs

*Example F-5. The fonts.c program (continued)*

```
    dpy = (Display *)xv_get(frame, XV_DISPLAY);
    font = (Xv_Font)xv_get(frame, XV_FONT);
    cur_font = (XFontStruct *)xv_get(font, FONT_INFO);
    xv_set(frame, FRAME_LEFT_FOOTER, xv_get(font, FONT_NAME), NULL);

    gcvalues.font = cur_font->fid;
    gcvalues.foreground = BlackPixel(dpy, DefaultScreen(dpy));
    gcvalues.background = WhitePixel(dpy, DefaultScreen(dpy));
    gcvalues.graphics_exposures = False;
    gc = XCreateGC(dpy, RootWindow(dpy, DefaultScreen(dpy)),
        GCForeground | GCBackground | GCFont | GCGraphicsExposures,
        &gcvalues);

    xv_main_loop(frame);
}

void
my_event_proc(win, event)
Xv_Window win;
Event *event;
{
    static int x = 10, y = 10;
    Window xwin = (Window)xv_get(win, XV_XID);
    char c;

    if (event_is_up(event))
        return;

    if (event_is_ascii(event)) {
        c = (char)event_action(event);
        XDrawString(dpy, xwin, gc, x, y, &c, 1);
        /* advance x to next position.  If over edge, linewrap */
        if ((x += XTextWidth(cur_font, &c, 1)) >= canvas_width) {
            y += cur_font->max_bounds.ascent +
                 cur_font->max_bounds.descent;
            x = 10;
        }
    } else if (event_action(event) == ACTION_SELECT) {
        x = event_x(event);
        y = event_y(event);
    } else if (event_action(event) == LOC_WINENTER)
        win_set_kbd_focus(win, xwin);
}

/*
 * check resizing so we know how wide to allow the user to type.
 */
void
my_resize_proc(canvas, width, height)
Canvas canvas;
int width, height;
{
    canvas_width = width;
}
```

*Example F-5. The fonts.c program (continued)*

```
void
change_font(item, value, event)
Panel_item   item;
Event       *event;
{
    static int  family, style, scale;
    char        buf[128];
    Frame       frame;
    char        *family_name;
    char        *style_name;
    int         scale_value;
    Xv_Font     font;

    frame = (Frame)xv_get(xv_get(item, PANEL_PARENT_PANEL), XV_OWNER);
    family_name = (char *)xv_get(family_item, PANEL_CHOICE_STRING,
                            xv_get(family_item, PANEL_VALUE));
    style_name = (char *)xv_get(style_item, PANEL_CHOICE_STRING,
                            xv_get(style_item, PANEL_VALUE));
    scale_value = (int) xv_get(scale_item, PANEL_VALUE);
    xv_set(frame, FRAME_BUSY, TRUE, NULL);
    font = (Xv_Font)xv_find(frame, FONT,
        FONT_FAMILY,    family_name,
        FONT_STYLE,     style_name,
        /* scale_value happens to coincide with Window_rescale_state */
        FONT_SCALE,     scale_value,
        /*
         * If run on a server that cannot rescale fonts, only font
         * sizes that exist should be passed
         */
        FONT_SIZES_FOR_SCALE, 12, 14, 16, 22,
        NULL);
    xv_set(frame, FRAME_BUSY, FALSE, NULL);

    if (!font) {
        if (item == family_item) {
            sprintf(buf, "cannot load '%s'", family_name);
            xv_set(family_item, PANEL_VALUE, family, NULL);
        } else if (item == style_item) {
            sprintf(buf, "cannot load '%s'", style_name);
            xv_set(style_item, PANEL_VALUE, style, NULL);
        } else {
            sprintf(buf, "Not available in %s scale.",
                xv_get(scale_item, PANEL_CHOICE_STRING, scale));
            xv_set(scale_item, PANEL_VALUE, scale, NULL);
        }
        xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
        return;
    }
    if (item == family_item)
        family = value;
    else if (item == style_item)
        style = value;
    else
        scale = value;
    cur_font = (XFontStruct *)xv_get(font, FONT_INFO);
    XSetFont(dpy, gc, cur_font->fid);
```

*Example F-5. The fonts.c program (continued)*

```
    sprintf(buf, "Current font: %s", xv_get(font, FONT_NAME));
    xv_set(frame, FRAME_LEFT_FOOTER, buf, NULL);
}

change_font_by_name(item, event)
Panel_item item;
Event *event;
{
    char buf[128];
    char *name = (char *)xv_get(item, PANEL_VALUE);
    Frame frame = (Frame)xv_get(xv_get(item, XV_OWNER), XV_OWNER);
    Xv_Font font;

    xv_set(frame, FRAME_BUSY, TRUE, NULL);
    font = (Xv_Font)font = (Xv_Font)xv_find(frame, FONT,
        FONT_NAME,      name,
        NULL);
    xv_set(frame, FRAME_BUSY, FALSE, NULL);

    if (!font) {
        sprintf(buf, "cannot load '%s'", name);
        xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);
        return PANEL_NONE;
    }
    cur_font = (XFontStruct *)xv_get(font, FONT_INFO);
    XSetFont(dpy, gc, cur_font->fid);
    sprintf(buf, "Current font: %s", xv_get(font, FONT_NAME));
    xv_set(frame, FRAME_LEFT_FOOTER, buf, NULL);
    return PANEL_NONE;
}
```

## F.6 x_draw.c

This program uses several Xlib drawing functions to draw various types of geometric objects. We integrate the XView color model (see Chapter 21, *Color*) to render each object in a different color.

*Example F-6. The x_draw.c program*

```
/*
 * x_draw.c -- demonstrates the use of Xlib drawing functions
 * inside an XView canvas.  Color is used, but not required.
 */
#include <xview/xview.h>
#include <xview/canvas.h>
#include <xview/cms.h>
#include <xview/xv_xrect.h>

/* indices into color table renders specified colors. */
#define WHITE   0
#define RED     1
#define GREEN   2
```

*Example F-6. The x_draw.c program (continued)*

```
#define BLUE    3
#define ORANGE  4
#define AQUA    5
#define PINK    6
#define BLACK   7

GC gc;                 /* GC used for Xlib drawing */
unsigned long *colors; /* the color table */

/*
 * initialize cms data to support colors specified above.  Assign
 * data to new cms -- use either static or dynamic cms depending
 * on -dynamic command line switch.
 */
main(argc, argv)
int     argc;
char    *argv[ ];
{
    static char stipple_bits[ ] = {0xAA, 0xAA, 0x55, 0x55};
    static Xv_singlecolor cms_colors[ ] = {
        { 255, 255, 255 },
        { 255, 0, 0 },
        { 0, 255, 0 },
        { 0, 0, 255 },
        { 250, 130, 80 },
        { 30, 230, 250 },
        { 230, 30, 250 },
    };
    Cms         cms;
    Frame       frame;
    Canvas      canvas;
    XFontStruct *font;
    Display     *display;
    XGCValues   gc_val;
    XID         xid;
    void        canvas_repaint();
    Xv_cmsdata  cms_data;
    int         use_dynamic = FALSE;

    /* Create windows */
    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);
    if (*++argv && !strcmp(*argv, "-dynamic"))
        use_dynamic = TRUE;

    frame = xv_create(NULL,FRAME,
        FRAME_LABEL,    "xv_canvas_x_draw",
        XV_WIDTH,       400,
        XV_HEIGHT,      300,
        NULL);

    cms = xv_create(NULL, CMS,
        CMS_SIZE,       7,
        CMS_TYPE,       use_dynamic? XV_DYNAMIC_CMS : XV_STATIC_CMS,
        CMS_COLORS,     cms_colors,
        NULL);
```

*Example F-6. The x_draw.c program (continued)*

```
    canvas = xv_create(frame, CANVAS,
        CANVAS_REPAINT_PROC,    canvas_repaint,
        CANVAS_X_PAINT_WINDOW,  TRUE,
/*      WIN_DYNAMIC_VISUAL,     use_dynamic,  */
      XV_VISUAL_CLASS, PseudoColor,
        WIN_CMS,                cms,
        NULL);

    /* Get display and xid */
    display = (Display *)xv_get(frame, XV_DISPLAY);
    xid = (XID)xv_get(canvas_paint_window(canvas), XV_XID);

    if (!(font = XLoadQueryFont(display, "fixed"))) {
        puts("cannot load fixed font");
        exit(1);
    }

    /* Create and initialize GC */
    gc_val.font = font->fid;
    gc_val.stipple =
        XCreateBitmapFromData(display, xid, stipple_bits, 16, 2);
    gc = XCreateGC(display, xid, GCFont | GCStipple, &gc_val);

    /* get the colormap from the canvas now that
     * the cms has been installed
     */
    colors = (unsigned long *)xv_get(canvas, WIN_X_COLOR_INDICES);

    /* Start event loop */
    xv_main_loop(frame);
}

/*
 * Draws onto the canvas using Xlib drawing functions.
 */
void
canvas_repaint(canvas, pw, display, xid, xrects)
Canvas          canvas;
Xv_Window       pw;
Display         *display;
Window          xid;
Xv_xrectlist    *xrects;
{
    static XPoint box[ ] = {
        {0,0}, {100,100}, {0,-100}, {-100,100}, {0,-100}
    };
    static XPoint points[ ] = {
        {0,0}, /* this point to be overwritten below */
        {25,0}, {25,0}, {25,0}, {25,0}, {-100,25},
        {25,0}, {25,0}, {25,0}, {25,0}, {-100,25},
        {25,0}, {25,0}, {25,0}, {25,0}, {-100,25},
        {25,0}, {25,0}, {25,0}, {25,0}, {-100,25},
        {25,0}, {25,0}, {25,0}, {25,0}, {-100,25},
    };

    XSetForeground(display, gc, colors[RED]);
```

*Example F-6. The x_draw.c program (continued)*

```
    XDrawString(display, xid, gc, 30, 20, "XFillRectangle", 14);
    XFillRectangle(display, xid, gc, 25, 25, 100, 100);
    XSetFunction(display, gc, GXinvert);
    XFillRectangle(display, xid, gc, 50, 50, 50, 50);
    XSetFunction(display, gc, GXcopy);

    XSetForeground(display, gc, colors[BLACK]);
    XDrawString(display, xid, gc, 155, 20, "XFillRect - stipple", 19);
    XSetFillStyle(display, gc, FillStippled);
    XFillRectangle(display, xid, gc, 150, 25, 100, 100);
    XSetFillStyle(display, gc, FillSolid);

    XSetForeground(display, gc, colors[BLUE]);
    XDrawString(display, xid, gc, 280, 20, "XDrawPoints", 11);
    points[0].x = 275; points[0].y = 25;
    XDrawPoints(display, xid, gc, points,
        sizeof(points)/sizeof(XPoint), CoordModePrevious);

    XSetForeground(display, gc, colors[ORANGE]);
    XDrawString(display, xid, gc, 30, 145, "XDrawLine - solid", 17);
    XDrawLine(display, xid, gc, 25, 150, 125, 250);
    XDrawLine(display, xid, gc, 25, 250, 125, 150);

    XSetForeground(display, gc, colors[AQUA]);
    XDrawString(display, xid, gc, 155, 145, "XDrawLine - dashed", 18);
    XSetLineAttributes(display, gc, 5,
        LineDoubleDash, CapButt, JoinMiter);
    XDrawLine(display, xid, gc, 150, 150, 250, 250);
    XDrawLine(display, xid, gc, 150, 250, 250, 150);
    XSetLineAttributes(display, gc, 0, LineSolid, CapButt, JoinMiter);

    XSetForeground(display, gc, colors[PINK]);
    XDrawString(display, xid, gc, 280, 145, "XDrawLines", 10);
    box[0].x = 275; box[0].y = 150;
    XDrawLines(display, xid, gc, box, 5, CoordModePrevious);

    XSetForeground(display, gc, colors[GREEN]);
    XDrawRectangle(display, xid, gc,
        5, 5, xv_get(pw, XV_WIDTH)-10, xv_get(pw, XV_HEIGHT)-10);
    XDrawRectangle(display, xid, gc,
        7, 7, xv_get(pw, XV_WIDTH)-14, xv_get(pw, XV_HEIGHT)-14);
}
```

## F.7 The Logo.c Module

In Chapter 25, *XView Internals*, the methods for writing XView extensions is discussed. Example F-7 contains all the functions outlined in the chapter. The chapter also contains listings of the header files required by this module.

*Example F-7. The Logo.c module*

```
/*
 * Logo.c -- a XView object class that paints an X logo in a window.
 * This object is subclassed from the window object to take advantage
 * of the window it creates.  This object has no attributes, so the
 * set and get functions are virtually empty.  The only internal
 * fields used by this object are a GC and a Pixmap.  The GC is used
 * to paint the Pixmap into the window.  The window object has no GC
 * associated with it or we would have inherited it.  This will
 * probably go away in the next version of XView.
 */
#include "logo_impl.h"
#include <xview/notify.h>
#include <xview/cms.h>
#include <X11/bitmaps/xlogo32>

/* declare the "methods" used by the logo class. */
static int logo_init(), logo_destroy();
static Xv_opaque logo_set(), logo_get();
static void logo_repaint();

Xv_pkg logo_pkg = {
    "Logo",                     /* package name */
    ATTR_PKG_UNUSED_FIRST,      /* package ID */
    sizeof(Logo_public),        /* size of the public struct */
    WINDOW,                     /* subclassed from the window package */
    logo_init,
    logo_set,
    logo_get,
    logo_destroy,
    NULL                        /* disable the use of xv_find() */
};

/* the only thing this object does is paint an X into its own window.
 * This is the event handling routine that is used to check for
 * Expose or Configure event requests.  the configure event clears
 * the window and the "expose" event causes a repaint of the X image.
 * The GC has its foreground and background colros set from the
 * CMS of the window from which this logo object is subclassed.
 */
static void
logo_redraw(logo_public, event)
Logo_public     *logo_public;
Event           *event;
{
    Logo_private *logo_private = LOGO_PRIVATE(logo_public);
    XEvent *xevent = event_xevent(event);

    if (xevent->xany.type == Expose && xevent->xexpose.count == 0) {
```

*Example F-7. The Logo.c module (continued)*

```
        Display *dpy = (Display *)xv_get(logo_public, XV_DISPLAY);
        Window window = (Window)xv_get(logo_public, XV_XID);
        int width = (int)xv_get(logo_public, XV_WIDTH);
        int height = (int)xv_get(logo_public, XV_HEIGHT);
        int x = (width - xlogo32_width)/2;
        int y = (height - xlogo32_height)/2;

        XCopyPlane(dpy, logo_private->bitmap, window, logo_private->gc,
            0, 0, xlogo32_width, xlogo32_height, x, y, 1L);
    } else if (xevent->xany.type == ConfigureNotify)
        XClearArea(xv_get(logo_public, XV_DISPLAY),
            xv_get(logo_public, XV_XID), 0, 0,
            xevent->xconfigure.width, xevent->xconfigure.height, True);
}

/* initialize the logo object -- create (alloc) an instance of it.
 * There are two parts to an object class: a public part and a private
 * part.  Each contains a pointer to the other, so link the two
 * together and initialize the remaining fields of the logo data
 * structure.  This includes creating the Xlogo pixmap.  However,
 * we do no initialize the logo's GC because it is dependent on its
 * window's cms and that isn't assigned to the window till the "set"
 * method.  See logo_set() below.
 */
static int
logo_init(owner, logo_public, avlist)
Xv_opaque       owner;
Logo_public     *logo_public;
Attr_avlist     avlist; /* ignored here */
{
    Logo_private *logo_private = xv_alloc(Logo_private);
    Display *dpy;
    Window win;

    if (!logo_private)
        return XV_ERROR;

    dpy = (Display *)xv_get(owner, XV_DISPLAY);
    win = (Window)xv_get(logo_public, XV_XID);

    /* link the public to the private and vice-versa */
    logo_public->private_data = (Xv_opaque)logo_private;
    logo_private->public_self = (Xv_opaque)logo_public;

    /* create the 1-bit deep pixmap of the X logo */
    if ((logo_private->bitmap = XCreatePixmapFromBitmapData(dpy, win,
        xlogo32_bits, xlogo32_width, xlogo32_height,
        1, 0, 1)) == NULL) {
        free(logo_private);
        return XV_ERROR;
    }
    /* set up event handlers to get resize and repaint events */
    xv_set(logo_public,
        WIN_NOTIFY_SAFE_EVENT_PROC,      logo_redraw,
        WIN_NOTIFY_IMMEDIATE_EVENT_PROC, logo_redraw,
        NULL);
```

*Example F-7. The Logo.c module (continued)*

```
    return XV_OK;
}

/* logo_set() -- the function called to set attributes in a logo
 * object.  This function is called when a logo is created after
 * the init routine as well as when the programmer calls xv_set.
 */
static Xv_opaque
logo_set(logo_public, avlist)
Logo_public *logo_public;
Attr_avlist avlist;
{
    Logo_private *logo_private = LOGO_PRIVATE(logo_public);
    Attr_attribute *attrs;

    for (attrs = avlist; *attrs; attrs = attr_next(attrs))
        switch ((int) attrs[0]) {
            case XV_END_CREATE : {
                /* this stuff *must* be here rather than in the "init"
                 * routine because the CMS is not loaded into the
                 * window object until the "set" routines are called.
                 */
                Cms cms = xv_get(logo_public, WIN_CMS);
                XGCValues gcvalues;
                Display *dpy =
                    (Display *)xv_get(logo_public, XV_DISPLAY);
                gcvalues.foreground =
                    x(unsigned long)v_get(cms, CMS_FOREGROUND_PIXEL);
                gcvalues.background =
                    (unsigned long)xv_get(cms, CMS_BACKGROUND_PIXEL);
                gcvalues.graphics_exposures = False;
                logo_private->gc = XCreateGC(dpy,
                    xv_get(logo_public, XV_XID),
                    GCForeground|GCBackground|GCGraphicsExposures,
                    &gcvalues);
            }
            default :
                xv_check_bad_attr(LOGO, attrs[0]);
                break;
        }

    return XV_OK;
}

/* logo_get() -- There are no logo attributes to get, so just return */
static Xv_opaque
logo_get(logo_public, status, attr, args)
Logo_public     *logo_public;
int             *status;
Attr_attribute  attr;
Attr_avlist     args;
{
    *status = xv_check_bad_attr(LOGO, attr);
    return (Xv_opaque)XV_OK;
}
```

*Example F-7. The Logo.c module (continued)*

```
/* destroy method: free the pixmap and the GC before freeing the object */
static int
logo_destroy(logo_public, status)
Logo_public     *logo_public;
Destroy_status   status;
{
    Logo_private *logo_private = LOGO_PRIVATE(logo_public);

    if (status == DESTROY_CLEANUP) {
        XFreePixmap(xv_get(logo_public, XV_DISPLAY),
            logo_private->bitmap);
        XFreeGC(xv_get(logo_public, XV_DISPLAY), logo_private->gc);
        free(logo_private);
    }

    return XV_OK;
}
```

Example Programs

## F.8 The Bitmap.c Module

In Chapter 25, *XView Internals*, the Bitmap package was introduced, but not fully listed. The following listing contains the entire Bitmap package implementation except for its header files (which are listed in the chapter).

*Example F-8. The Bitmap.c module*

```
/*
 * Bitmap.c -- an XView object class that displays an arbitrary
 * pixmap.  This is similar to the Logo object, but the programmer
 * may specify the bitmap to use via the BITMAP_FILE attribute.
 */
#include "bitmap_impl.h"
#include <xview/notify.h>
#include <xview/cms.h>
#include <X11/Xutil.h>

/* declare the "methods" used by the bitmap class. */
static int bitmap_init(), bitmap_destroy();
static Xv_opaque bitmap_set(), bitmap_get();
static void bitmap_repaint();

Xv_pkg bitmap_pkg = {
    "Bitmap2",              /* package name */
    ATTR_PKG_BITMAP,        /* package ID */
    sizeof(Bitmap_public),  /* size of the public struct */
    WINDOW,                 /* subclassed from the window package */
    bitmap_init,
    bitmap_set,
    bitmap_get,
    bitmap_destroy,
    NULL                    /* disable the use of xv_find() */
```

*Example F-8. The Bitmap.c module (continued)*

```
};

static void
bitmap_redraw(bitmap_public, event)
Bitmap_public     *bitmap_public;
Event            *event;
{
    Bitmap_private *bitmap_private = BITMAP_PRIVATE(bitmap_public);
    XEvent *xevent = event_xevent(event);

    if (bitmap_private->bitmap &&
        xevent->xany.type == Expose && xevent->xexpose.count == 0) {
        Display *dpy = (Display *)xv_get(bitmap_public, XV_DISPLAY);
        Window window = (Window)xv_get(bitmap_public, XV_XID);
        int width = (int)xv_get(bitmap_public, XV_WIDTH);
        int height = (int)xv_get(bitmap_public, XV_HEIGHT);
        int x = (width - bitmap_private->width)/2;
        int y = (height - bitmap_private->height)/2;

        XCopyPlane(dpy, bitmap_private->bitmap, window,
            bitmap_private->gc, 0, 0, bitmap_private->width,
            bitmap_private->height, x, y, 1L);
    } else if (xevent->xany.type == ConfigureNotify)
        XClearArea(xv_get(bitmap_public, XV_DISPLAY),
            xv_get(bitmap_public, XV_XID), 0, 0,
            xevent->xconfigure.width, xevent->xconfigure.height,
            True);
}

/* initialize the bitmap object by creating (alloc) an instance
 * of it.  There are two parts to an object class: a public part
 * and a private part.  Each contains a pointer to the other, so
 * link the two together and initialize the remaining fields of
 * the bitmap data structure.  Do no initialize the bitmap's GC
 * because it is dependent on its window's cms and that isn't
 * assigned to the window till the "set" method.  Also, wait till
 * till the "set" method to initialize the bitmap file specified.
 */
static int
bitmap_init(owner, bitmap_public, avlist)
Xv_opaque       owner;
Bitmap_public     *bitmap_public;
Attr_avlist     avlist; /* ignored here */
{
    Bitmap_private *bitmap_private = xv_alloc(Bitmap_private);

    if (!bitmap_private)
        return XV_ERROR;

    /* link the public to the private and vice-versa */
    bitmap_public->private_data = (Xv_opaque)bitmap_private;
    bitmap_private->public_self = (Xv_opaque)bitmap_public;

    /* set up event handlers to get resize and repaint events */
    xv_set(bitmap_public,
        WIN_NOTIFY_SAFE_EVENT_PROC,      bitmap_redraw,
```

*Example F-8. The Bitmap.c module (continued)*

```
        WIN_NOTIFY_IMMEDIATE_EVENT_PROC, bitmap_redraw,
        NULL);

    return XV_OK;
}

/* bitmap_set() -- the function called to set attributes in a bitmap
 * object.  This function is called when a bitmap is created after
 * the init routine as well as when the programmer calls xv_set.
 */
static Xv_opaque
bitmap_set(bitmap_public, avlist)
Bitmap_public *bitmap_public;
Attr_avlist avlist;
{
    Bitmap_private *bitmap_private = BITMAP_PRIVATE(bitmap_public);
    Attr_attribute *attrs;

    for (attrs = avlist; *attrs; attrs = attr_next(attrs))
        switch ((int) attrs[0]) {
            case BITMAP_FILE : {
                int val, x, y;
                Display *dpy =
                    (Display *)xv_get(bitmap_public, XV_DISPLAY);
                Window window =
                    (Window)xv_get(bitmap_public, XV_XID);
                Pixmap old = bitmap_private->bitmap;
                if (XReadBitmapFile(dpy, window, attrs[1],
                    &bitmap_private->width, &bitmap_private->height,
                    &bitmap_private->bitmap, &x, &y) != BitmapSuccess)
                {
                    xv_error(bitmap_public,
                        ERROR_STRING, "Unable to load bitmap file",
                        ERROR_PKG,    BITMAP,
                        NULL);
                    bitmap_private->bitmap = old;
                }
                break;
            }
            case BITMAP_PIXMAP :
                xv_error(bitmap_public,
                    ERROR_CANNOT_SET, attrs[0],
                    ERROR_PKG,        BITMAP,
                    NULL);
                break;
            case XV_END_CREATE : {
                /* this stuff *must* be here rather than in the "init"
                 * routine because the CMS is not loaded into the
                 * window object until the "set" routines are called.
                 */
                Cms cms = xv_get(bitmap_public, WIN_CMS);
                XGCValues gcvalues;
                Display *dpy =
                    (Display *)xv_get(bitmap_public, XV_DISPLAY);
                gcvalues.foreground =
                    (unsigned long)xv_get(cms, CMS_FOREGROUND_PIXEL);
```

Example Programs

*Example F-8. The Bitmap.c module (continued)*

```
                gcvalues.background =
                    (unsigned long)xv_get(cms, CMS_BACKGROUND_PIXEL);
                gcvalues.graphics_exposures = False;
                bitmap_private->gc =
                    XCreateGC(dpy, xv_get(bitmap_public, XV_XID),
                        GCForeground|GCBackground|GCGraphicsExposures,
                        &gcvalues);
            }
            default :
                xv_check_bad_attr(BITMAP, attrs[0]);
                break;
        }
    return XV_OK;
}

static Xv_opaque
bitmap_get(bitmap_public, status, attr, args)
Bitmap_public     *bitmap_public;
int             *status;
Attr_attribute  attr;
Attr_avlist     args;
{
    Bitmap_private *bitmap_private = BITMAP_PRIVATE(bitmap_public);

    switch ((int) attr) {
        case BITMAP_PIXMAP :
            return (Xv_opaque)bitmap_private->bitmap;
        case BITMAP_FILE : /* can't get this attribute */
        default :
            *status = xv_check_bad_attr(BITMAP, attr);
            return (Xv_opaque)XV_OK;
    }
}

/* destroy method: free the pixmap and the GC before freeing object */
static int
bitmap_destroy(bitmap_public, status)
Bitmap_public     *bitmap_public;
Destroy_status   status;
{
    Bitmap_private *bitmap_private = BITMAP_PRIVATE(bitmap_public);

    if (status == DESTROY_CLEANUP) {
        if (bitmap_private->bitmap)
            XFreePixmap(xv_get(bitmap_public, XV_DISPLAY),
                bitmap_private->bitmap);
        XFreeGC(xv_get(bitmap_public, XV_DISPLAY),
            bitmap_private->gc);
        free(bitmap_private);
    }
    return XV_OK;
}
```

## F.9 The panel_dnd.c Program

In Chapter 7, *Panels*, a Drop Target Items is described. This program creates and uses a Panel Drop Target Item.

*Example F-9. The panel_dnd.c program*

```
/*
 * panel_dnd.c - provides text fields, a textsw, and
 * several drop targets to demonstate ways to receive
 * and illustrate drag and drop operations.
 */

#include <malloc.h>
#include <xview/xview.h>
#include <xview/dragdrop.h>
#include <xview/panel.h>
#include <xview/svrimage.h>
#include <xview/textsw.h>

static unsigned short normal_bitmap[ ] = {
#include "normal.icon"
};

static unsigned short busy_bitmap[ ] = {
#include "busy.icon"
};

static unsigned short normal2_bitmap[ ] = {
#include "normal2.icon"
};

static unsigned short busy2_bitmap[ ] = {
#include "busy2.icon"
};

static char * dnd_codes[7 ] = {
    "OK",
    "Error",
    "Illegal Target",
    "Timeout",
    "Unable to obtain selection",
    "Dropped on root window",
    "*** Unknown return code"
};

Frame frame;
Panel panel;
Panel_item drop_target[3 ];
Drag_drop dnd;


/*ARGSUSED*/
static void
hide_drop_targets(item, event)
    Panel_item item;
```

Example Programs

*Example F-9. The panel_dnd.c program (continued)*

```
    Event *event;
{
    int         i;
    int         show;

    show = !xv_get(drop_target[0], XV_SHOW);
    for (i=0; i<=2; i++)
      xv_set(drop_target[i],
             XV_SHOW, show,
             0);
    xv_set(item,
      PANEL_LABEL_STRING, show ? "Hide drop targets" : "Show drop targets",
      0);
}


/*ARGSUSED*/
static void
inactivate_drop_targets(item, event)
    Panel_item item;
    Event *event;
{
    int         i;
    int         inactive;

    inactive = !xv_get(drop_target[0], PANEL_INACTIVE);
    for (i=0; i<=2; i++)
      xv_set(drop_target[i],
             PANEL_INACTIVE, inactive,
             0);
    xv_set(item,
      PANEL_LABEL_STRING, inactive ? "Activate drop targets" :
          "Inactivate drop targets",
      0);
}


static void
get_primary_selection(sel_req)
    Selection_requestor sel_req;
{
    long        length;
    int              format;
    char       *sel_string;
    char       *string;

    xv_set(sel_req, SEL_TYPE, XA_STRING, 0);
    string = (char *) xv_get(sel_req, SEL_DATA, &length, &format);
    if (length != SEL_ERROR) {
      /* Create a NULL-terminated version of 'string' */
      sel_string = (char *) xv_calloc(1, length+1);
      strncpy(sel_string, string, length);
      /* Print out primary selection string */
      printf("Primary selection=
    } else
      printf("*** Unable to get primary selection.0);
```

*Example F-9. The panel_dnd.c program (continued)*

```
}


static int
drop_target_notify_proc(item, value, event)
    Panel_item       item;
    unsigned int    value;
    Event      *event;
{
    Selection_requestor sel_req;

    printf("(drop_target_notify_proc) %s action= ",
      xv_get(item, PANEL_LABEL_STRING));
    sel_req = xv_get(item, PANEL_DROP_SEL_REQ);
    switch (event_action(event)) {
      case ACTION_DRAG_COPY:
      printf("ACTION_DRAG_COPY,0);
      get_primary_selection(sel_req);
      break;
      case ACTION_DRAG_MOVE:
      printf("ACTION_DRAG_MOVE0);
      get_primary_selection(sel_req);
      break;
      case LOC_DRAG:
      printf("LOC_DRAG, result= %s0, dnd_codes[MIN(value, 6)]);
      break;
      default:
      printf("%d0, event_action(event));
      break;
    }
    return XV_OK;
}


main(argc, argv)
    int argc;
    char **argv;
{
    Server_image    busy_glyph[2];
    Server_image    normal_glyph[2];
    Panel       panel2;

    xv_init(XV_INIT_ARGS, argc, argv, 0);

    frame = xv_create(NULL, FRAME,
      FRAME_LABEL, "Drag and Drop Test",
      0);
    panel = xv_create(frame, PANEL,
      PANEL_LAYOUT, PANEL_VERTICAL,
      0);
    xv_create(panel, PANEL_BUTTON,
      PANEL_LABEL_STRING, "Inactivate Drop Targets",
      PANEL_NOTIFY_PROC, inactivate_drop_targets,
      0);
    xv_create(panel, PANEL_BUTTON,
      PANEL_LABEL_STRING, "Hide Drop Targets",
```

*Example F-9. The panel_dnd.c program (continued)*

```
        PANEL_NOTIFY_PROC, hide_drop_targets,
        0);
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_STRING, "Text field #1:",
        PANEL_VALUE, "Hello world!",
        PANEL_VALUE_DISPLAY_LENGTH, 20,
        0);
    xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_STRING, "Text field #2:",
        PANEL_VALUE_DISPLAY_LENGTH, 20,
        0);
    dnd = xv_create(panel, DRAGDROP, 0);
    xv_create(dnd, SELECTION_ITEM,
        SEL_DATA, "dnd selection data",
        0);
    normal_glyph[0] = xv_create(NULL, SERVER_IMAGE,
        XV_HEIGHT, 64,
        XV_WIDTH, 64,
        SERVER_IMAGE_DEPTH, 1,
        SERVER_IMAGE_BITS, normal_bitmap,
        0);
    busy_glyph[0] = xv_create(NULL, SERVER_IMAGE,
        XV_HEIGHT, 64,
        XV_WIDTH, 64,
        SERVER_IMAGE_DEPTH, 1,
        SERVER_IMAGE_BITS, busy_bitmap,
        0),
    normal_glyph[1] = xv_create(NULL, SERVER_IMAGE,
        XV_HEIGHT, 64,
        XV_WIDTH, 64,
        SERVER_IMAGE_DEPTH, 1,
        SERVER_IMAGE_BITS, normal2_bitmap,
        0);
    busy_glyph[1] = xv_create(NULL, SERVER_IMAGE,
        XV_HEIGHT, 64,
        XV_WIDTH, 64,
        SERVER_IMAGE_DEPTH, 1,
        SERVER_IMAGE_BITS, busy2_bitmap,
        0),
    drop_target[0] = xv_create(panel, PANEL_DROP_TARGET,
        PANEL_LABEL_STRING, "Full Drop Target:",
        PANEL_NOTIFY_PROC, drop_target_notify_proc,
        PANEL_DROP_DND, dnd,
        PANEL_DROP_FULL, TRUE,
        PANEL_DROP_GLYPH, normal_glyph[0],
        PANEL_DROP_BUSY_GLYPH, busy_glyph[0],
        0);
    xv_create(panel, PANEL_DROP_TARGET,
        PANEL_LABEL_STRING, "Full Drop Target #2:",
        PANEL_NOTIFY_PROC, drop_target_notify_proc,
        PANEL_DROP_DND, dnd,
        PANEL_DROP_FULL, TRUE,
        PANEL_DROP_GLYPH, normal_glyph[1],
        PANEL_DROP_BUSY_GLYPH, busy_glyph[1],
        0);
    drop_target[1] = xv_create(panel, PANEL_DROP_TARGET,
```

*Example F-9. The panel_dnd.c program (continued)*

```
        PANEL_LABEL_STRING, "Default Empty Drop Target:",
        PANEL_NOTIFY_PROC, drop_target_notify_proc,
        PANEL_DROP_SITE_DEFAULT, TRUE,
        0);
    if (xv_get(drop_target[1], PANEL_DROP_SITE_DEFAULT) != TRUE) {
        printf("PANEL_DROP_SITE_DEFAULT failed0);
        exit(1);
    }
    drop_target[2] = xv_create(panel, PANEL_DROP_TARGET,
        PANEL_LABEL_STRING, "Default Full Drop Target:",
        PANEL_NOTIFY_PROC, drop_target_notify_proc,
        PANEL_DROP_DND, dnd,
        PANEL_DROP_FULL, TRUE,
        0);
    window_fit(panel);

    xv_create(frame, TEXTSW,
        WIN_ROWS, 4,
        WIN_COLUMNS, 10,
        0);
    panel2 = xv_create(frame, PANEL,
        PANEL_LAYOUT, PANEL_VERTICAL,
        0);
    xv_create(panel2, PANEL_MESSAGE,
        PANEL_LABEL_STRING, "New panel",
        0);
    xv_create(panel2, PANEL_TEXT,
        PANEL_LABEL_STRING, "Text field:",
        PANEL_VALUE_DISPLAY_LENGTH, 20,
        0);
    window_fit(panel2);

    window_fit(frame);

    xv_main_loop(frame);
    exit(0);
}
```

Example Programs