19
Drag and Drop

This chapter describes the XView packages that support drag and drop. *Drag and drop* allows data to be moved within an application or between applications. Using drag and drop, objects are selected, then moved or copied to a new location. The selected object is the drag and drop *source*. Selecting the source is referred to as *sourcing the drag*. An area within the source application or within another application *receives the drop* in a pre-registered *drop-site*.

This chapter describes packages that are new for XView Version 3. The function `xv_decode_drop()` that supports dragging and dropping is now obsolete. Note that `xv_decode_drop()` will *not* work correctly once an application is compiled for V3. The function is still provided for backwards compatibility for XView Version 2 programs.

Many OPEN LOOK applications use drag and drop operations. For example, an application may allow text in a text subwindow to be sourced and copied or moved to a new location within the text subwindow. Similarly, a File Manager's file icon can be selected and dragged to the Print Tool icon for printing. OPEN LOOK defines many uses for drag and drop, including loading files into an application. Figure 19-1 shows a file being dragged into an editor and Figure 19-2 shows the application loading the file (receiving the drop).

Drag and drop supports *drop previewing* where the drop-site image changes to show that it is a valid drop-site. It also supports *drag feedback* where the pointer image changes to indicate that an item is being dragged. The data transfer for an XView drag and drop operation is normally implemented using selections. You need to be familiar with the selection package to use drag and drop. The selection package is described in Chapter 18, *Selections*. Note that the operation of drag and drop using OPEN LOOK differs from selections in the following ways:

- Drag and drop is a single gesture rather than a series of separate actions.
- Drag and drop operates on complete pre-registered data objects. Using drag and drop, the application decides which areas may be selected, and then the user may select and drag the area.

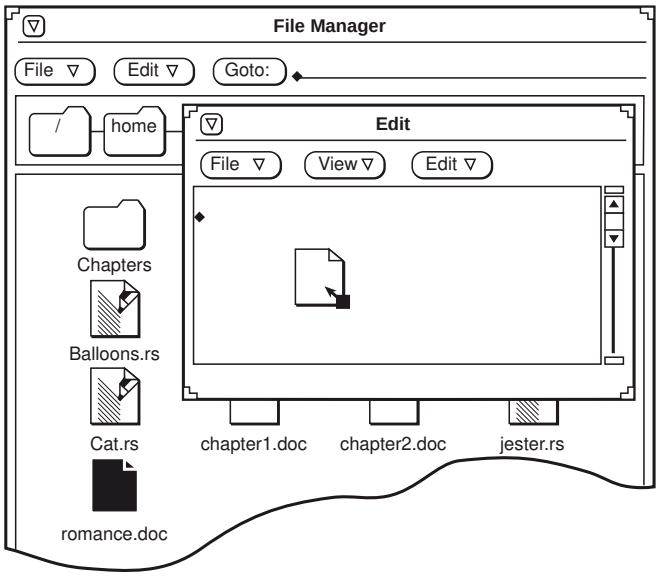

Figure 19-1. Dragging a file onto an application

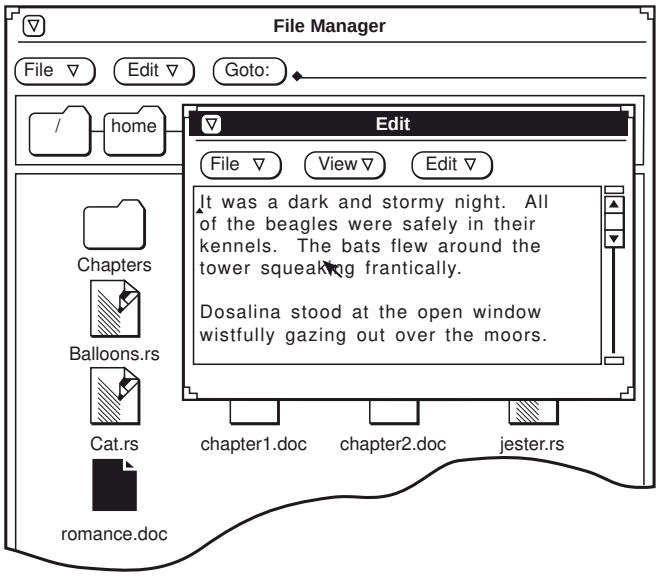

Figure 19-2. Loading a file by dropping

19.1 Drag and Drop Objects

Drag and drop is facilitated using two objects: `DRAGDROP` and `DROP_SITE_ITEM`; a `DRAGDROP` object represents the source of the drag. A `DROP_SITE_ITEM` object indicates a destination that is a valid drop-site. All applications need to include the file *<xview/dragdrop.h>* to use these packages. All `DRAGDROP` attributes are prefixed by `DND` and all `DROP_SITE_ITEM` attributes are prefixed by `DROP_SITE`. Both Figure 19-3 and Figure 19-4 show the class hierarchy for these objects.

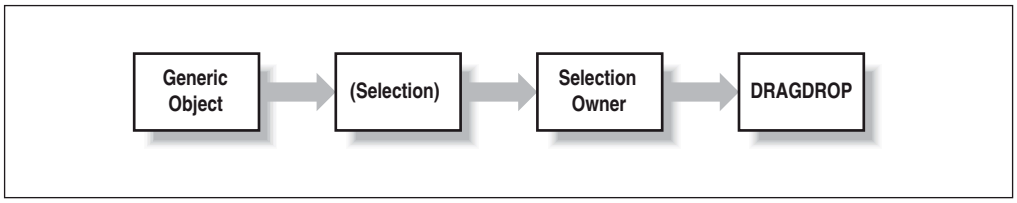

Figure 19-3. DRAGDROP class hierarchy

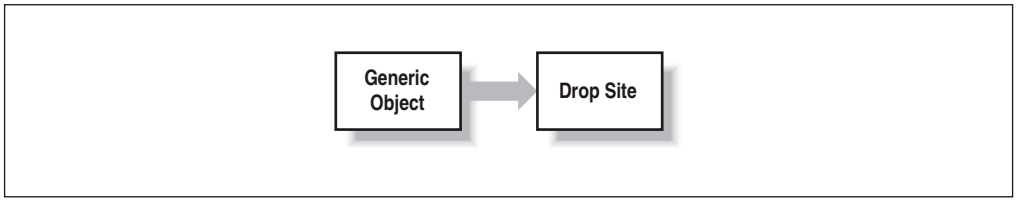

Figure 19-4. DROP_SITE_ITEM class hierarchy

Drag and drop operations involve three steps which the following sections describe:

- The application that is to receive the drop must register a region or several regions as valid drop-sites. This lets the application inform other applications that it is interested in receiving drops.
- The application containing the source for the drag and drop determines the point at which to initiate the drag operation. For example, a drag operation could be initiated when text or a file icon is selected and dragged. A `DRAGDROP` object is created by the source application. The drag begins when the application calls the procedure `dnd_send_drop()`.
- An application that has registered a drop-site must be prepared to receive a drop.

19.2 Registering Drop-sites

An application that is to receive a drop must register a region or several regions as valid drop-sites. This allows drag sources to determine the validity of potential drop-sites and enables the application to receive the events for previewing and dropping. For example, a graphic icon or any other window may be registered as a drop-site. The drop-sites may consist of one region or of several regions described by one or more rectangles within the application. If an application does not register an area as a drop-site, it cannot receive or preview drops.

An object instantiated from the `DROP_SITE_ITEM` package describes a drop-site. A drop-site item is subclassed from the `GENERIC` package and is not a visible object. The following example demonstrates how to make the rectangle representing an entire window a drop-site:

```
drop_site = (Xv_drop_site)xv_create(window, DROP_SITE_ITEM,
              DROP_SITE_ID,          1234,
              DROP_SITE_REGION,      xv_get(window, WIN_RECT),
              DROP_SITE_EVENT_MASK,  DND_ENTERLEAVE | DND_MOTION,
              DROP_SITE_DEFAULT,     TRUE,
              NULL);
```

Instances of `DROP_SITE_ITEM` are owned by window-based objects. The attribute `DROP_SITE_ID` is an uninterpreted ID that may be used by the application to distinguish one drop-site from the next. This attribute is useful when more than one drop-site is set on an object (see Section 19.2.2, “Handling Events,” for details on using this attribute). Section 19.2.1, “Adding and Deleting Regions,” describes regions and the attribute `DROP_SITE_REGION`. The attribute `DROP_SITE_EVENT_MASK` is a mask set on the drop-site item. It is used to specify if the region(s) within the drop-site should receive preview events (see Section 19.2.2, “Handling Events,” for more details).

`DROP_SITE_DEFAULT` establishes a default drop-site for drops forwarded from the window manager. Such drops include drops on icons and window manager decorations. Only one drop-site default should be specified per base frame; specifying more than one will have unpredictable results. Setting this attribute provides only a hint to the window manager. The drop-site default is used most often as a way to have drops forwarded from iconified applications.

19.2.1 Adding and Deleting Regions

Individual drop-site regions should correspond to those objects within the window that may receive drops. For example, if there is a 64x64 icon at (10,10) within a canvas that is an acceptable drop-site, then a drop-site item should be created. The drop-site item’s region should be set to a rectangle with coordinates at (10,10) and a size of 64x64. If the position of the objects within the window change or the objects are clipped by other objects, *it is the responsibility of the application* to update the drop-site item to correspond to the new position of these items. For example, if an object is deleted, its region should be removed. If an object is being clipped by its containing window border, or a sibling window border, and is no longer viewable, its region should be removed or updated to reflect the viewable portion of the drop-site. If an object moves, relative to the base frame, the drop-site item region

needs to be updated by deleting the old region and adding the new, updated region. Drop-sites obscured by other application windows need not be updated. When the window that owns a drop-site is unmapped, the drop-sites it owns are also unmapped (made "invisible") to drop sources. They are returned when the window is mapped again.

If an application does not keep the region information up-to-date, users may be given false indication that *old* drop-site areas can receive drops.

If a drop-site item's owner is destroyed with `xv_destroy()`, any drop-site regions attached to it are also destroyed.

The attribute `DROP_SITE_REGION` associates a region to a drop-site item. `DROP_SITE_REGION` *appends* regions to any existing regions within the drop-site item. The coordinates in the `rect` should be relative to the drop-site item's owner's window. An `xv_get()` of a region returns a pointer to an allocated `Rect *` structure. This should be freed using `xv_free()` once the application has finished using it.

The attribute `DROP_SITE_REGION_PTR` is similar to `DROP_SITE_REGION` except that it accepts a `NULL`-terminated array of regions. It appends to any existing regions within the drop-site item. A `NULL rect` is defined to be one with width or height equal to 0.

The `DROP_SITE_DELETE_REGION` attribute removes a region from the drop-site item. When a `NULL` is passed as a value for this attribute, all regions in the drop-site are removed.

The `DROP_SITE_DELETE_REGION_PTR` attribute removes a list of regions from the drop-site item. Passing a `NULL` as an argument removes all regions in the drop-site. Typically, when a region needs to be updated, `xv_set()` should be called with two attributes: `DROP_SITE_DELETE_REGION_PTR` with a `NULL` value, followed by `DROP_SITE_REGION_PTR`. Note that the order of these two attributes is important.

19.2.2 Handling Events

The drop-site item's owner is a window. The event procedure for the owner window receives the drag and drop action events and handles them accordingly. Drag and drop semantic events are shown in Table 19-1.

Table 19-1. Drag and Drop Semantic Events

Event	Purpose
`ACTION_DRAG_PREVIEW`	Preview event.
`ACTION_DRAG_MOVE`	Move the source data to the drop-site window.
`ACTION_DRAG_COPY`	Copy the source data to the drop-site window.

19.2.2.1 Preview events

The attribute DROP_SITE_EVENT_MASK sets a mask on the drop-site item to specify if the region(s) within the drop-site should receive preview events. Preview events indicate the state of the mouse relative to the drop-site. Preview events are indicated when event_action() returns ACTION_DRAG_PREVIEW (in this case, event_id() returns either LOC_WINENTER or LOC_WINEXIT). These events cue the drop-site when the mouse enters or leaves a drop-site region. Drop-sites can also select for LOC_DRAG events that are sent as the mouse moves across a region. These events are delivered to the *event procedure of the drop-site item's owner*, (the pointer coordinates are contained within the event). The coordinates are in the drop-site owner's coordinate space.

DROP_SITE_EVENT_MASK is only a hint since there is no guarantee the source of the drag will send previewing events. When preview events arrive, an application may invert the image of the drop-site. This or some other change that indicates the area is an acceptable drop-site.

An uninterpreted ID specified with the attribute DROP_SITE_ID may be used by the application program to distinguish one drop-site from the next. This attribute is useful when more than one drop-site has been set on an object. This ID is sent along with the ACTION_DRAG_PREVIEW, ACTION_DRAG_MOVE, and ACTION_DRAG_COPY events. If no DROP_SITE_ID is set, the package creates a unique ID for each drop-site region.

19.2.2.2 Event forwarding

The procedure dnd_is_forwarded() lets an application determine if a drop or preview event is forwarded from some other site. This may happen when the user drops on the window manager's decoration window, or on a icon when the application is iconified (only if DROP_SITE_DEFAULT is set). The corresponding drop or preview event will have its DND_FORWARDED flag set.

In general, if an application handles previewing, it should check to see if the preview event was forwarded. If the event was forwarded, it should *not* invert or highlight the drop-site.

19.2.2.3 Handling drop and preview events

A preview event is sent to the drop-site owner when the user is dragging over a registered drop-site. A drop event is sent to the drop-site owner when the user is dragging over a registered drop-site and releases the mouse buttons. There are two ways that the drop-site owner can handle such events in order to preview or to receive the drop:

- The window can set a WIN_EVENT_PROC and define a callback procedure to handle the drop.
- The window can use a notification interposing routine.

If your application uses an interposing routine, two types of interposition procedures must be registered for receiving both drop and previewing events. Both a safe, NOTIFY_SAFE, and an immediate, NOTIFY_IMMEDIATE, interposer needs to be registered. If both safe and immedi-

ate interposers are not registered, there are cases when events may be delayed in arriving at the drop-site and may not arrive “on time.”

19.3 Sourcing the Drag

The application containing the source for the drag and drop determines the point to initiate the drag operation. For example, a drag operation could be initiated when text or a file icon is selected and dragged. What events determine the initiation point are entirely up to the application programmer.

Sometime before, or when an application initiates a drag and drop operation it needs to create a drag and drop object and set its attributes. An instance of a DRAGDROP object is created using xv_create():

```
dnd_object = xv_create(owner, DRAGDROP,
      <attribute-value list>,
      NULL)
```

A DRAGDROP object’s owner is an Xv_window and the *<attribute-value list>* may contain drag and drop attributes (DND prefix) or since the DND package is subclassed from selection owner package, any of the SELECTION_OWNER attributes. For example, the drag and drop object dnd below uses a selection attribute:

```
dnd = (Xv_drag_drop)xv_create(window, DRAGDROP,
          DND_TYPE,           DND_COPY
          DND_X_CURSOR,       arrow_cursor,
          DND_ACCEPT_CURSOR,  drop_here_cursor,
          SEL_CONVERT_PROC,   SelectionConvert,
          NULL);
```

The owner of the drag and drop object should be the window in which the drag operation is initiated from. Since the DRAGDROP object is used as part of the selection transaction that implements the drag and drop operation, the selection transaction must also be completed before the DRAGDROP object can be reused.

NOTE

A drag and drop object can only be used in one drag and drop operation at a time.

The attribute DND_TYPE defines whether the drag and drop operation will be a copy or a move. This is just a hint to the destination. If the type is a DND_MOVE operation and if the destination honors the hint, the destination asks the source to convert the DELETE target. The DND_ACCEPT_CURSOR attribute, and other cursor related attributes are described in the Section 19.3.3, “Defining the Drag/Accept Cursor.”

Once the DRAGDROP object is created, it can be used to initiate a drag and drop operation by calling dnd_send_drop().

19.3.1 Initiating the Drop Operation

To initiate a drag operation, after the drag and drop object is created, an application calls `dnd_send_drop()`. This procedure is defined by XView and is responsible for changing the root cursor, sending previewing events to appropriate drop-sites and sending a *trigger* message, an event of type `ACTION_DRAG_COPY` or `ACTION_DRAG_MOVE`, to the drop-site that is to receive the drop. In sequence, `dnd_send_drop()` does the following:

- Creates a unique selection if this is required.
- Grabs the pointer and changes the cursor.
- Sends preview events to possible drop-sites.
- Sends the *kicker* message to the drop-site to initiate the transfer.
- Insures that the recipient acknowledges the drop.
- Returns a status.

For example, an application such as `filemgr` calls this routine when the user drags the mouse for a number of pixels over a selected file icon.

The form for `dnd_send_drop()` is:

```
int
dnd_send_drop(object)
   Dnd  object;
```

Dnd is a drag and drop object: `dnd_send_drop()` does not return until the user releases all of the mouse buttons, drops the object, or hits the STOP key. The procedure `dnd_send_drop()` returns one of the values shown in Table 19-2.

Table 19-2. dnd_send_drop() Return Values

Return Value	Description
`DND_ILLEGAL_TARGET`	The user dropped on an object that has not registered interest in drag and drop.
`DND_ROOT`	The user dropped on the root window.
`DND_SELECTION`	A unique selection could not be obtained.
`DND_TIMEOUT`	The destination did not respond to the kicker message (the drop).
`DND_ABORTED`	The user aborted the drag operation by hitting the STOP key.
`XV_OK`	The drag and drop operation has begun.
`XV_ERROR`	An unexpected error occurred.

19.3.2 Interaction with the Selection Package

Using drag and drop, data is transferred through selections. Since the DRAGDROP package is subclassed from the SELECTION_OWNER package, it fully supports selections. A selection is associated with the drag and drop object which allows the drop-site item's owner to obtain the data through selection transactions. The selection is normally a transient selection that persists only for the duration of the drag and drop operation. A transient selection should be used even if the dragged object is already associated with a selection such as the primary selection (PRIMARY). If the primary selection were used, and another application acquired the primary selection during the drag and drop transfer, the drag and drop operation would fail.

The application can use the SEL_RANK attribute to associate a selection with the drag and drop object. If the application does not own a selection (SEL_OWN) at the time dnd_send_drop() is called, XView creates a transient selection and associates it with the drag and drop object. This selection can be obtained by using xv_get() on the drag and drop object with the SEL_RANK attribute. *It is the responsibility of the application to disown a selection associated with the drag and drop object when it has completed the operation*. This is regardless of whether the application or the toolkit initially established ownership of the selection.

If XView created a transient selection and dnd_send_drop() failed (did not return XV_OK), the toolkit will disown the selection.

Since the source will be required to reply to requests from the destination for data conversion, the source should have either a conversion procedure or selection items set on the drag and drop object *before* it calls dnd_send_drop().

19.3.3 Defining the Drag/Accept Cursor

The cursor changes when an object is being dragged. The cursor used to indicate dragging an object is set with DND_CURSOR. This attribute changes the pointer used during the drag portion of the drag and drop operation. The default drag cursor is the predefined OPEN LOOK drag cursor. DND_X_CURSOR is an alternative to DND_CURSOR; it accepts an XID of a cursor instead of an Xv_object.

The attribute DND_ACCEPT_CURSOR defines the special "accept" cursor that is used when the mouse is over an acceptable drop-site. The default value for this attribute is a predefined OPEN LOOK drag and drop cursor. DND_ACCEPT_X_CURSOR is an alternative way to set the accept cursor. This attribute accepts an XID of a cursor instead of an Xv_cursor.*

The cursor package also provides special support for drag and drop cursors for text. Refer to Chapter 13, *Cursors*, for information on these special drag and drop cursors.

*Future implementations of this package will also support a DND_REJECT_CURSOR that may be used when drop-sites are currently unreceptive to drops.

19.3.4 Timeout Value

The attribute DND_TIMEOUT_VALUE defines the amount of time to wait for an acknowledgment from the drop destination after the kicker message has been sent to the source (ACTION_DRAG_COPY or ACTION_DRAG_MOVE).

19.4 Receiving a Drop

An application that has registered a drop-site may receive a drop. A window or object that owns the drop-site item may receive one of two events when a valid drop-site is dropped on. An ACTION_DRAG_COPY indicates a copy and an ACTION_DRAG_MOVE indicates a move. These events are the *trigger* message that indicate that a drop has happened.

Upon receiving a drop event, the application should call dnd_decode_drop() and pass in as a parameter the ACTION_DRAG_MOVE or ACTION_DRAG_COPY event that it received along with an instantiated SELECTION_REQUESTOR object. The SELECTION_REQUESTOR object is used since drag and drop data transfers use the selection package to facilitate the transfer.

The dnd_decode_drop() function initiates a data transfer using the selection mechanism. It decodes the drop event as follows:

- It associates the selection rank defined within the drop event with the selection object that was passed into dnd_decode_drop().
- It sends an acknowledgment to the source of the drag and drop, informing it that the transaction has begun.
- It returns the drop-site item that was dropped on if dnd_decode_drop() could begin the drag and drop transaction or returns XV_ERROR if it fails to initiate the drag and drop transaction. This may happen when the selection defined within the drop event does not exist.

The form of the procedure dnd_decode_drop() is:

```
Xv_drop_site
dnd_decode_drop(sel_object, drop_event)
     Selection_requestor    sel_object;
     Event                 *drop_event;
```

If dnd_decode_drop() returns a drop-site item, it is the responsibility of the application to continue the drag and drop transaction. This occurs by making selection requests on the selection object passed into dnd_decode_drop(). See Chapter 18, *Selections*, for details on selection transfers.

The data transfer mechanism for sending and receiving drop data is not limited to using selections. Any mechanism could be used for the transfer, including: the file system, sockets, or one of the other *Alternate Transport Mechanisms.* Typically, such an alternate transfer mechanism would be initiated after the initial selection request.

The macro `dnd_is_local(drop_event)` returns `TRUE` if the source and destination of the drag and drop are the same application. This macro is available so that local (occurring within a single client) drag and drops can be optimized.

19.4.0.1 The move operation

If the application receives an `ACTION_DRAG_MOVE`, it should simulate a move by performing a copy followed by a delete. The delete is initiated by the requestor asking the holder to convert the `DELETE` target. For example, a blocking selection transfer uses the following code:

```
xv_set(sel_object, SEL_TYPE_NAME, "DELETE", NULL);
(void)xv_get(sel_object, SEL_DATA, &length, &format);
```

A non-blocking selection transfer would be initiated as follows:

```
xv_set(sel_object, SEL_TYPE_NAME, "DELETE", NULL);
sel_post_request(sel_object);
```

The application should only ask the owner to delete the selection if the drop event indicates a move operation *and* the application determines it is appropriate to do so. This will typically happen after the application has *successfully* transferred the data. To avoid data loss, it is very important to assure that the data was successfully transferred to the destination application before the data is deleted from the application that is the source of the drag.

19.4.0.2 The done procedure

At the end of the drag and drop operation, after the data has been transferred, the application received the drop must inform XView that the drag and drop operation has been completed. It does so by calling `dnd_done()`:

```
dnd_done(sel_object);
   Selection_requestor sel_object;
```

19.5 Sample Program—Sourcing a Drag

Example 19-1 shows a program that sources a drag. This program and Example 19-2 show how to implement a drop-site item and receive a drop.

Example 19-1. Sourcing a drag

```
/*
 * source1.c - Example of how to source a drag and drop operation.
 *
 */

#include <stdio.h>
#include <sys/types.h>
#include <xview/xview.h>
#include <xview/frame.h>
#include <xview/canvas.h>
#include <xview/cursor.h>
#include <xview/dragdrop.h>
#include <xview/sel_pkg.h>
#include <xview/xv_xrect.h>
#include <xview/svrimage.h>
#include <X11/Xlib.h>
#include <X11/Xatom.h>

short drop_icon[ ] = {
#include "./drop.icon"
};

#define  POINT_IN_RECT(px, py, rx, ry, rw, rh) \
               ((px) >= rx && (py) >= ry && \
               (px) < rx+rw && (py) < ry+rh)

#define  STRING_MSG  "chromosome: DNA-containing body of the nucleus."

#define  HOST   0
#define  STRING 1
#define  LENGTH 2

Frame          frame;
Canvas    canvas;
Dnd       dnd;
Cursor         arrow_cursor;
Server_image   arrow_image;
Server_image   arrow_image_mask;
Server_image   box_image;
Server_image   drop_here_image;
Cursor         drop_here_cursor;
Selection_owner sel;
Selection_item  selItem[5];
Atom      selAtom[5];

int  SelectionConvert();
extern int sel_convert_proc();

static XColor   fg = {0L, 65535, 65535, 65535};
```

Example 19-1. Sourcing a drag (continued)

```
static XColor   bg = {0L, 0, 0, 0};

typedef struct _DragObject {
    Server_image  image;
    int           x, y;
    unsigned int  w, h;
    int           inverted;
} DragObject;

DragObject  dO;

main(argc, argv)
    int    argc;
    char **argv;
{
    void        EventProc(),
          SelectionLose(),
          PaintCanvas();
    Xv_Server   server;
    Cursor arror_cursor;

    server = xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);

    frame = xv_create((Window)NULL, FRAME,
                XV_LABEL,       "Drag & Drop Source",
                XV_X,                10,
                XV_Y,                10,
                FRAME_SHOW_FOOTER,   True,
                0);

    canvas = xv_create(frame, CANVAS,
                        XV_HEIGHT,        100,
                        XV_WIDTH,         300,
                        CANVAS_REPAINT_PROC,    PaintCanvas,
                        CANVAS_X_PAINT_WINDOW,  TRUE,
                        0);

    xv_set(canvas_paint_window(canvas),
                        WIN_BIT_GRAVITY,  ForgetGravity,
                WIN_CONSUME_EVENTS,
                     WIN_MOUSE_BUTTONS,
                     LOC_DRAG,
                               WIN_RESIZE,
                          0,
                        WIN_EVENT_PROC,   EventProc,
                        0);
                /* Create the drag cursor images */
    arrow_image = xv_create(NULL, SERVER_IMAGE,
                SERVER_IMAGE_BITMAP_FILE, "arrow.bm", NULL);
    arrow_image_mask = xv_create(NULL, SERVER_IMAGE,
                SERVER_IMAGE_BITMAP_FILE, "arrow_mask.bm", NULL);
                /* Create a cursor to use in dnd ops. */
    arrow_cursor = XCreatePixmapCursor(XV_DISPLAY_FROM_WINDOW(canvas),
                (XID)xv_get(arrow_image, XV_XID),
                (XID)xv_get(arrow_image_mask, XV_XID),
                &fg, &bg, 61, 3);
```

Example 19-1. Sourcing a drag (continued)

```
    drop_here_image = xv_create(NULL, SERVER_IMAGE,
                XV_WIDTH,       64,
                XV_HEIGHT,      64,
                SERVER_IMAGE_BITS,   drop_icon,
                0);

    drop_here_cursor = XCreatePixmapCursor(
                               XV_DISPLAY_FROM_WINDOW(canvas),
                (XID)xv_get(drop_here_image, XV_XID),
                (XID)xv_get(drop_here_image, XV_XID),
                &fg, &bg, 32, 32);

    dO.image = xv_create(NULL, SERVER_IMAGE,
                SERVER_IMAGE_BITMAP_FILE, "arrowb.bm", NULL);
    dO.w = xv_get(dO.image, XV_WIDTH);
    dO.h = xv_get(dO.image, XV_HEIGHT);
    dO.inverted = False;

    CreateSelection(server, canvas_paint_window(canvas));

    window_fit(frame);

    xv_main_loop(frame);
    exit(0);

}
CreateSelection(server, window)
    Xv_Server   server;
{
    char   name[15];
    int       len;

          /* Primary selection, acquired whenever the arrow
           * bitmap is selected by the user.
           * This is not a requirement for dnd to work.
           */
    sel = xv_create(window, SELECTION_OWNER,
                SEL_RANK,       XA_PRIMARY,
                SEL_LOSE_PROC,       SelectionLose,
                0);

                /* Create the drag and drop object.  */
    dnd = xv_create(window, DRAGDROP,
                DND_TYPE,       DND_COPY,
                DND_X_CURSOR,        arrow_cursor,
                DND_ACCEPT_X_CURSOR, drop_here_cursor,
                SEL_CONVERT_PROC,    SelectionConvert,
                0);

                /* Associate some selection items with the dnd object.*/
    (void) gethostname(name, 15);
    selAtom[HOST] = (Atom)xv_get(server, SERVER_ATOM, "HOST_NAME");
    selItem[HOST] = xv_create(dnd, SELECTION_ITEM,
                SEL_TYPE,       selAtom[HOST],
                SEL_DATA,       (Xv_opaque)name,
```

Example 19-1. Sourcing a drag (continued)

```
                0);

    selAtom[STRING] = (Atom)XA_STRING;
    selItem[STRING] = xv_create(dnd, SELECTION_ITEM,
                SEL_TYPE,       selAtom[STRING],
                SEL_DATA,       (Xv_opaque)STRING_MSG,
                0);

    len = strlen(STRING_MSG);
    selAtom[LENGTH] = (Atom)xv_get(server, SERVER_ATOM, "LENGTH");
    selItem[LENGTH] = xv_create(dnd, SELECTION_ITEM,
                SEL_TYPE,       selAtom[LENGTH],
                SEL_FORMAT,          sizeof(int)*NBBY,
                SEL_LENGTH,          1,
                SEL_DATA,       (Xv_opaque)&len,
                0);
}

void
EventProc(window, event)
Xv_Window       window;
Event           *event;
{
    static int drag_pixels = 0;
    static int dragging = False;

    switch (event_action(event)) {
      case ACTION_SELECT:
     if (event_is_down(event)) {
         dragging = False;
                /* If the user selected our dnd object, highlight
                 * the box and acquire the primary selection.
                 */
         if (POINT_IN_RECT(event_x(event), event_y(event),
                           dO.x, dO.y, dO.w, dO.h)) {
          xv_set(sel, SEL_OWN, True, 0);
          dO.inverted = True;
          PaintObject(dO, xv_get(window, XV_XID),
                                XV_DISPLAY_FROM_WINDOW(window));
         } else
          /* If the user selected outside of the dnd object,
           * de-highlight the object. And release the primary
           * selection.
           */
          xv_set(sel, SEL_OWN, False, 0);
     } else
         drag_pixels = 0;
       break;
      case LOC_DRAG:
                /* If the user dragged at least five pixel over our
                 * dnd object, begin the dnd operation.
                 */
       if (event_left_is_down(event)) {
        if (POINT_IN_RECT(event_x(event),
                    event_y(event),dO.x,dO.y,dO.w,dO.h))
          dragging = True;
```

Example 19-1. Sourcing a drag (continued)

```
            if (dragging && drag_pixels++ == 5) {
             xv_set(frame, FRAME_LEFT_FOOTER, "Drag and Drop:", 0);
             switch (dnd_send_drop(dnd)) {
            case XV_OK:
                xv_set(frame, FRAME_LEFT_FOOTER,
                          "Drag and Drop: Began", 0);
                break;
               case DND_TIMEOUT:
                xv_set(frame, FRAME_LEFT_FOOTER,
                          "Drag and Drop: Timed Out",0);
                break;
               case DND_ILLEGAL_TARGET:
                xv_set(frame, FRAME_LEFT_FOOTER,
                          "Drag and Drop: Illegal Target",0);
                break;
               case DND_SELECTION:
                xv_set(frame, FRAME_LEFT_FOOTER,
                          "Drag and Drop: Bad Selection",0);
                break;
               case DND_ROOT:
                xv_set(frame, FRAME_LEFT_FOOTER,
                          "Drag and Drop: Root Window",0);
                break;
               case XV_ERROR:
                xv_set(frame, FRAME_LEFT_FOOTER,
                          "Drag and Drop: Failed",0);
                break;
             }
            drag_pixels = 0;
            }
     }
        break;
    }
}

PaintObject(object, win, dpy)
    DragObject   object;
    Window win;
    Display     *dpy;
{
    static GC   gc;
    static int  gcCreated = False;

    if (!gcCreated) {
        XGCValues gcv;
        gcv.stipple = (Pixmap) xv_get(object.image, XV_XID);
        gcv.fill_style = FillStippled;
        gc = XCreateGC(dpy, win, GCStipple|GCForeground|GCBackground|
                                          GCFillStyle, &gcv);
        XSetForeground(dpy, gc, BlackPixel(dpy, XDefaultScreen(dpy)));
        XSetBackground(dpy, gc, WhitePixel(dpy, XDefaultScreen(dpy)));
    }

    if (object.inverted) {
     XSetFillStyle(dpy, gc, FillSolid);
```

Example 19-1. Sourcing a drag (continued)

```
     XDrawRectangle(dpy, win, gc, object.x-1, object.y-1, 66, 66);
     XSetFillStyle(dpy, gc, FillStippled);
    } else
     XClearWindow(dpy, win);

    XSetTSOrigin(dpy, gc, object.x, object.y);
    XFillRectangle(dpy, win, gc, object.x, object.y, 65, 65);
}

void
PaintCanvas(canvas, paint_window, dpy, xwin, xrects)
    Canvas        canvas;         /* unused */
    Xv_Window     paint_window;   /* unused */
    Display      *dpy;
    Window        xwin;
    Xv_xrectlist *xrects;         /* unused */
{
    unsigned    width, height;
    int         x, y;

    width = xv_get(paint_window, XV_WIDTH);
    height = xv_get(paint_window, XV_HEIGHT);

    x = (width/2)-(dO.w/2);
    y = (height/2)-(dO.h/2);

    dO.x = x;
    dO.y = y;

    PaintObject(dO, xwin, dpy);
}
/* The convert proc is called whenever someone makes a request
 * to the dnd selection.  Two cases we handle within the convert
 * proc: DELETE and _SUN_SELECTION_END.  Everything else we pass
 * on to the default convert proc which knows about our selection
 * items.
 */
int
SelectionConvert(seln, type, data, length, format)
    Selection_owner   seln;
    Atom        *type;
    Xv_opaque         *data;
    long        *length;
    int               *format;
{
    Xv_Server         server = XV_SERVER_FROM_WINDOW(xv_get(seln,
                                      XV_OWNER));

    if (*type == (Atom)xv_get(server,
                      SERVER_ATOM, "_SUN_SELECTION_END")) {
          /* Destination has told us it has completed the drag
          * and drop transaction.  We should respond with a
          * zero-length NULL reply.
          */
      xv_set(dnd, SEL_OWN, False, 0);
      xv_set(frame, FRAME_LEFT_FOOTER, "Drag and Drop: Completed",0);
```

Example 19-1. Sourcing a drag (continued)

```
     *format = 32;
     *length = 0;
     *data = NULL;
     *type = (Atom)xv_get(server, SERVER_ATOM, "NULL");
     return(True);
    } else if (*type == (Atom)xv_get(server,
                                    SERVER_ATOM, "DELETE")) {
                /* Destination asked us to delete the selection.
                 * If it is appropriate to do so, we should.
                 */
     *format = 32;
     *length = 0;
     *data = NULL;
     *type = (Atom)xv_get(server, SERVER_ATOM, "NULL");
     return(True);
    } else
                /* Let the default convert procedure deal with the
                 * request.
                 */
     return(sel_convert_proc(seln, type, data, length, format));
}

/* When we lose the primary selection, this procedure is called.
 * We dehigh-light our selection.
 */
void
SelectionLose(seln)
    Selection_owner seln;
{
    Xv_Window   owner = xv_get(seln, XV_OWNER);

    if (xv_get(seln, SEL_RANK) == XA_PRIMARY) {
        dO.inverted = False;
        PaintObject(dO, xv_get(owner,
                          XV_XID), XV_DISPLAY_FROM_WINDOW(owner));
    }
}
```

19.6 Sample Program—Drop Site Item and Destination

Example 19-2 provides a example of a program that creates a drop-site item and sets up to receive a drop.

Example 19-2. A drop-site item example

```
/*
 * dest.c - Example of how to register interest in receiving
 *          drag and drop events and how to complete a drag
 *          and drop operation.
 *
 */
#include <stdio.h>
```

Example 19-2. A drop-site item example (continued)

```
#include <xview/xview.h>
#include <xview/frame.h>
#include <xview/canvas.h>
#include <xview/panel.h>
#include <xview/font.h>
#include <xview/dragdrop.h>
#include <xview/xv_xrect.h>
#include <X11/Xlib.h>

#define  DROP_WIDTH      65
#define  DROP_HEIGHT     65

#define  BULLSEYE_SITE      1

Frame         frame;
Canvas        canvas;
Panel         panel;
Xv_drop_site  drop_site;
Server_image  drop_image;
Server_image  drop_image_inv;
Panel_item    p_string,
              p_length,
              p_host;

Selection_requestor  sel;

int      inverted;

main(argc, argv)
    int argc;
    char **argv;
{
    void      EventProc(),
             PaintCanvas(),
             ResizeCanvas();
    Xv_Font      font;

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

    frame = xv_create((Window)NULL, FRAME,
                      XV_X,       330,
                      XV_Y,       10,
                      XV_WIDTH,   10,
                      XV_LABEL,   "Drag & Drop Destination",
                      NULL);


    panel = xv_create(frame, PANEL,
                      PANEL_LAYOUT,  PANEL_VERTICAL,
                      NULL);

    p_string = xv_create(panel, PANEL_TEXT,
                      PANEL_LABEL_STRING,       "Dropped Text:",
                      PANEL_VALUE_DISPLAY_LENGTH, 50,
                      NULL);
    p_host = xv_create(panel, PANEL_TEXT,
```

Example 19-2. A drop-site item example (continued)

```
                        PANEL_LABEL_STRING,       "From host:",
                        PANEL_VALUE_DISPLAY_LENGTH, 15,
                        NULL);
    p_length = xv_create(panel, PANEL_TEXT,
                        PANEL_LABEL_STRING,       "Length:",
                        PANEL_VALUE_DISPLAY_LENGTH, 6,
                        NULL);

    window_fit(panel);

    canvas = xv_create(frame, CANVAS,
                        XV_HEIGHT,      100,
                        XV_WIDTH,       WIN_EXTEND_TO_EDGE,
                        XV_X,           0,
                            WIN_BELOW,  panel,
                            CANVAS_REPAINT_PROC,    PaintCanvas,
                            CANVAS_RESIZE_PROC,     ResizeCanvas,
                            CANVAS_X_PAINT_WINDOW,  TRUE,
                            NULL );

    xv_set(canvas_paint_window(canvas),
                            WIN_BIT_GRAVITY,        ForgetGravity,
                            WIN_CONSUME_EVENTS,
                                        WIN_RESIZE,
                                        NULL,
                            WIN_EVENT_PROC,  EventProc,
                            NULL);

    drop_image = xv_create(NULL, SERVER_IMAGE,
                            SERVER_IMAGE_BITMAP_FILE, "./bullseye.bm",
                            NULL);

    drop_image_inv = xv_create(NULL, SERVER_IMAGE,
                            SERVER_IMAGE_BITMAP_FILE, "./bullseyeI.bm",
                            NULL);

           /* Selection requestor object that will be
           * passed into dnd_decode_drop() and later used
           * to make requests to the source of the
           * drop.
           */
    sel = xv_create(canvas, SELECTION_REQUESTOR, NULL);

           /* This application has one drop site with
           * site id BULLSEYE_SITE and whose shape will
           * be described by a rectangle.  If
           * animation is supported, it would like to
           * receive LOC_DRAG, LOC_WINENTER and
           * LOC_WINEXIT events.
           */
    drop_site = xv_create(canvas_paint_window(canvas), DROP_SITE_ITEM,
                            DROP_SITE_ID,           BULLSEYE_SITE,
                            DROP_SITE_EVENT_MASK,   DND_ENTERLEAVE,
                            NULL);

    inverted = False;
```

Example 19-2. A drop-site item example (continued)

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

void
EventProc(window, event)
    Xv_Window       window;
    Event           *event;
{
    switch (event_action(event)) {
           /* When drop previewing is available, if
           * the drop site has selected for previewing
           * events (DROP_SITE_EVENT_MASK) then it will
           * receive ACTION_DRAG_PREVIEW events from
           * the source as requested.
           */
      case ACTION_DRAG_PREVIEW:
       switch(event_id(event)) {
         case LOC_WINENTER:
           inverted = True;
           break;
         case LOC_WINEXIT:
           inverted = False;
           break;
         case LOC_DRAG:
           break;
       }
       PaintCanvas(NULL, window, XV_DISPLAY_FROM_WINDOW(window),
                  xv_get(window, XV_XID), NULL);
       break;
      case ACTION_DRAG_COPY:
      case ACTION_DRAG_MOVE: {
       Xv_drop_site  ds;
       Xv_Server  server = XV_SERVER_FROM_WINDOW(event_window(event));

           /* If the user dropped over an acceptable
           * drop site, the owner of the drop site will
           * be sent an ACTION_DROP_{COPY, MOVE} event.
           */
           /* To acknowledge the drop and to associate the
           * rank of the source's selection to our
           * requestor selection object, we call
           * dnd_decode_drop().
           */
       if ((ds = dnd_decode_drop(sel, event)) != XV_ERROR) {

           if (xv_get(ds, DROP_SITE_ID) == BULLSEYE_SITE)
              UpdatePanel(server, sel);

            /* If this is a move operation, we must ask
            * the source to delete the selection object.
            * We should only do this if the transfer of
            * data was successful.
            */
```

Example 19-2. A drop-site item example (continued)

```
            if (event_action(event) == ACTION_DRAG_MOVE) {
               int length, format;

                 xv_set(sel, SEL_TYPE_NAME, "DELETE", NULL);
                 (void)xv_get(sel, SEL_DATA, &length, &format);
            }

             /* To complete the drag and drop operation,
             * we tell the source that we are all done.
             */
            dnd_done(sel);
            inverted = False;
            PaintCanvas(NULL, window, XV_DISPLAY_FROM_WINDOW(window),
                       xv_get(window, XV_XID), NULL);

       } else
            printf ("drop error\n");
         break;
       }
       default:
         break;
    }
}


UpdatePanel(server, sel)
    Xv_Server              server;
    Selection_requestor       sel;
{
    int     length,
            format,
           *string_length;
    char    buf[7],
           *string,
           *hostname;

    xv_set(sel, SEL_TYPE, XA_STRING, NULL);
    string = (char *)xv_get(sel, SEL_DATA, &length, &format);
    if (length != SEL_ERROR) {
           xv_set(p_string, PANEL_VALUE, string, NULL);
           free (string);
    }

    xv_set(sel,
            SEL_TYPE, xv_get(server, SERVER_ATOM, "LENGTH"), NULL);
    string_length = (int *)xv_get(sel, SEL_DATA, &length, &format);
    if (length != SEL_ERROR) {
       sprintf(buf, "%d", *string_length);
        xv_set(p_length, PANEL_VALUE, buf, NULL);
        free ((char *)string_length);
    }

    xv_set(sel, SEL_TYPE, xv_get(server, SERVER_ATOM, "HOST_NAME"), NULL);
    hostname = (char *)xv_get(sel, SEL_DATA, &length, &format);
    if (length != SEL_ERROR) {
           xv_set(p_host, PANEL_VALUE, hostname, NULL);
```

Example 19-2. A drop-site item example (continued)

```
            free (hostname);
    }
    xv_set(sel, SEL_TYPE_NAME, "_SUN_SELECTION_END", NULL);
    (void)xv_get(sel, SEL_DATA, &length, &format);
}

void
PaintCanvas(canvas, paint_window, dpy, xwin, xrects)
    Canvas        canvas;         /* unused */
    Xv_Window     paint_window;   /* unused */
    Display      *dpy;
    Window        xwin;
    Xv_xrectlist *xrects;         /* unused */
{
    static GC   gc;
    static int  gcCreated = False;
    static int  lastMode = False;
    int         width, height;
    int         x, y;
    Rect       *r;

    if (!gcCreated) {
        XGCValues gcv;
        gcv.stipple = (Pixmap) xv_get(drop_image, XV_XID);
        gcv.foreground = BlackPixel(dpy, XDefaultScreen(dpy));
        gcv.background = WhitePixel(dpy, XDefaultScreen(dpy));
        gcv.fill_style = FillStippled;
        gc = XCreateGC(dpy, xwin, GCStipple|GCForeground|GCBackground|
                                  GCFillStyle, &gcv);
    }

    if (lastMode != inverted) {
       if (!inverted)
           XSetStipple(dpy, gc, (Pixmap) xv_get(drop_image, XV_XID));
       else
           XSetStipple(dpy, gc, (Pixmap) xv_get(drop_image_inv,
                                                XV_XID));
       lastMode = inverted;
    }

    width = xv_get(paint_window, XV_WIDTH);
    height = xv_get(paint_window, XV_HEIGHT);

    x = (width/2)-(DROP_WIDTH/2);
    y = (height/2)-(DROP_HEIGHT/2);

    XClearArea(dpy, xwin, x, y, DROP_WIDTH, DROP_HEIGHT, False);
    XSetTSOrigin(dpy, gc, x, y);
    XFillRectangle(dpy, xwin, gc, x, y, DROP_WIDTH, DROP_HEIGHT);
}

void
ResizeCanvas(canvas, width, height)
    Canvas        canvas;
    int          width;
    int          height;
```

Example 19-2. A drop-site item example (continued)

```
{
    int        x, y;
    Rect       rect;

    x = (width/2)-(DROP_WIDTH/2);
    y = (height/2)-(DROP_HEIGHT/2);

    rect.r_left = x;
    rect.r_top = y;
    rect.r_width = DROP_WIDTH;
    rect.r_height = DROP_HEIGHT;

                    /* Update the drop site information. */
    xv_set(drop_site, DROP_SITE_DELETE_REGION_PTR, NULL,
                   DROP_SITE_REGION, &rect,
                   NULL);
}
```

19.7 Drag and Drop Package Summary

Table 19-3 lists the procedures and macros used for dragging and dropping. Table 19-4 lists the `DROP_SITE_ITEM` and `DRAGDROP` package attributes. These attributes, procedures, and macros are described fully in the *XView Reference Manual*.

Table 19-3. DROP_SITE_ITEM and DRAGDROP Procedures and Macros

`dnd_decode_drop()`
`dnd_done()`
`dnd_is_forwarded()`
`dnd_is_local()`
`dnd_send_drop()`

Table 19-4. DROP_SITE_ITEM and DRAGDROP Attributes

DROP_SITE_ITEM Attributes	DRAGDROP Attributes
`DROP_SITE_DEFAULT`	`DND_ACCEPT_CURSOR`
`DROP_SITE_DELETE_REGION`	`DND_ACCEPT_X_CURSOR`
`DROP_SITE_DELETE_REGION_PTR`	`DND_CURSOR`
`DROP_SITE_EVENT_MASK`	`DND_TIMEOUT_VALUE`
`DROP_SITE_ID`	`DND_TYPE`
`DROP_SITE_REGION`	`DND_X_CURSOR`
`DROP_SITE_REGION_PTR`	