7
Panels

The XView PANEL package implements the OPEN LOOK *control area*. Panels are used in many different contexts—property sheets, notices, and menus all use panels in their implementation. The main function of a panel is to manage a variety of *panel items*. Figure 7-3 shows examples of panel items from the *OPEN LOOK GUI*. Because some panel items may not contain windows that handle their own events, the PANEL package is responsible for propagating events to the appropriate panel item. This chapter addresses issues specific to panels, the management of panel items and the distribution of events to those items. We look at basic issues common to all panel items before introducing each of the eight different panel item packages. Finally, we look at several advanced topics regarding panel usage.

The PANEL package is subclassed from the WINDOW package. In typical usage, you create a panel and set certain panel-specific attributes. Figure 7-1 shows the class hierarchy for panels and Figure 7-2 shows the class hierarchy for a panel item.

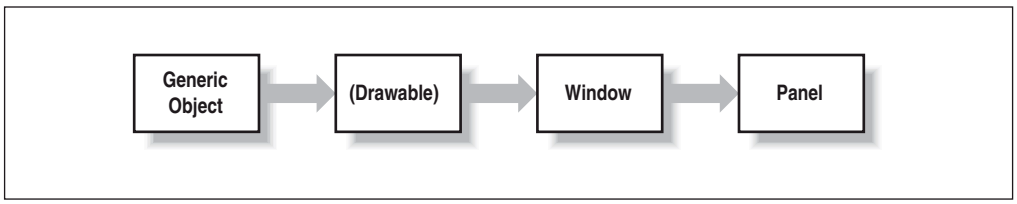

Figure 7-1. Panel package class hierarchy

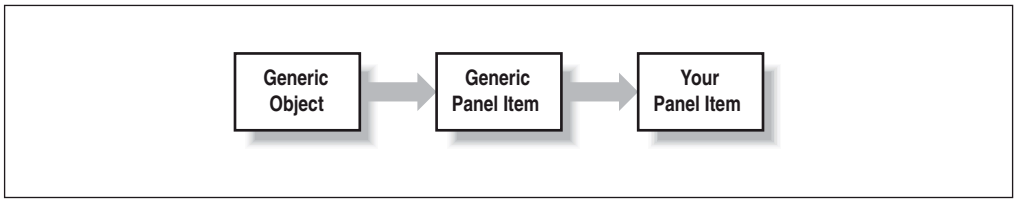

Figure 7-2. Panel item class hierarchy

Panels set up and manage event-handling masks and routines for themselves and their panel items. The application does not set event masks or install an event callback routine unless it needs to track events above and beyond what the PANEL package does by default (typical applications will not need to do this). Even so, this is probably better accomplished via interposing functions as discussed in Chapter 20, *The Notifier*.

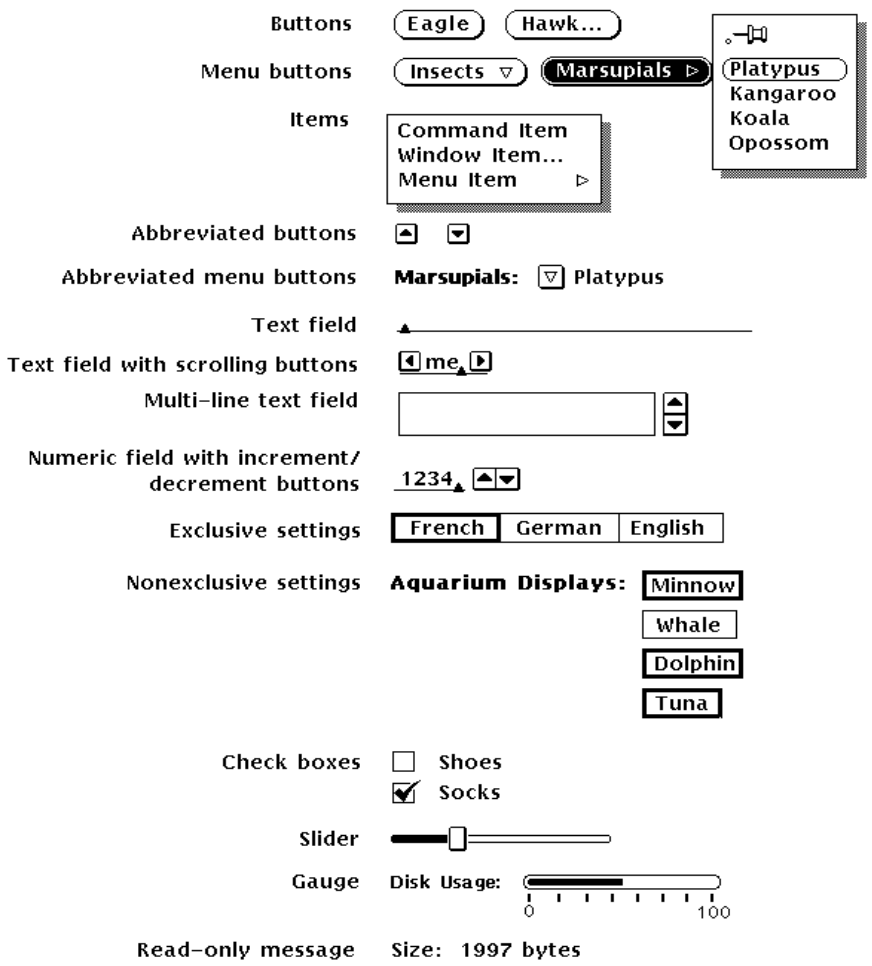

*Figure 7-3. Controls in an OPEN LOOK GUI implementation**

* XView supports the scrolling list item but it is not shown here.

The PANEL package handles all the repainting and resizing events automatically. Panels are not used to display graphics, so there is no need to capture repaint events. Rather than deal with other events specifically callback routines are not installed on panels, but set for each panel item. Because of the varying types of panel items, each item's callback function may be invoked by a different action from the user. While clicking on a panel button is all that is necessary to activate the button's callback routine, a text panel item might be configured to call its notification callback routine when the user presses the RETURN key.

Since panel items express interest in different events, it is the responsibility of the PANEL package to track all events within the panel's window and dispatch events to the proper panel item depending on its type. In some cases, if an event happens over a certain panel item and that item is not interested in that event, the event may be sent to another panel item. For example, what happens if a key is pressed over a panel button? Because the panel button has no interest in the event, the panel will send the event to a text panel item, if one exists elsewhere in the panel.

Section 7.19, "Advanced Panel Usage," describes panel event handling and repainting.

7.1 Creating a Panel

You create a panel by calling xv_create() and specifying the PANEL package and a NULL-terminated list of attribute-value pairs. A panel must be created as a child of the frame. All programs that use panels or panel items must include *<xview/panel.h>*. Because panels are uninteresting without panel items, Example 7-1 shows how to create a simple frame, a panel and a panel button. Selecting the panel button causes the program to exit. This is the same *quit.c* program as shown in Chapter 3, *Creating XView Applications*.

Example 7-1. The quit.c program

```
/*
 * quit.c -- simple program to display a panel button that says "Quit".
 * Selecting the panel button exits the program.
 */
#include <xview/xview.h>
#include <xview/frame.h>
#include <xview/panel.h>

Frame frame;

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

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

    frame = (Frame)xv_create (NULL, FRAME,
        FRAME_LABEL,    argv[0],
        XV_WIDTH,       200,
```

Example 7-1. The quit.c program (continued)

```
        XV_HEIGHT,      100,
        NULL);

    panel = (Panel)xv_create (frame, PANEL, NULL);

    (void) xv_create (panel, PANEL_BUTTON,
            PANEL_LABEL_STRING,         "Quit",
            PANEL_NOTIFY_PROC,          quit,
            NULL);

    xv_main_loop (frame);
    exit(0);
}

void
quit()
{
    xv_destroy_safe(frame);
}
```

7.1.0.1 Fonts and panels

OPEN LOOK is somewhat restrictive about the use of fonts within panels. Many panel items cannot have their fonts changed at all. Of those that can, none can have their fonts set individually.* However, if a font is set in the panel itself, it is then inherited by the panel items whose fonts can be changed. This guarantees the consistency of fonts in panels. In addition, a panel's font may only be specified when the panel is created.

7.1.1 Scrollable Panels

Scrollable panels are not OPEN LOOK-compliant, but are provided for historical reasons. They are basically just like panels, except that typically not all panel items are in view. A vertical scrollbar attached to the panel allows the user to navigate to the panel items desired. Again, because this type of interface is not OPEN LOOK-compliant, you are discouraged from using this package.

In order to deal with the complications involved with attaching a scrollbar to a panel, the scrollable panel package is subclassed from the `CANVAS` package and thus, the `OPENWIN` package. Scrollable panels are created the same way panels are, but the package name to use is `SCROLLABLE_PANEL`. The scrollable panel package does not create the scrollbars, however. You must create them separately:

*The exact list of panel items varies and may change. Currently, it includes the text used in the value of panel text items.

```
Scrollable_panel sp;
Scrollbar sb;

sp = xv_create(frame, SCROLLABLE_PANEL, NULL);
sb = xv_create(sp, SCROLLBAR, NULL);
```

The principle difference between canvases and scrollable panels is the management of events and the existence of panel items. In canvases, the programmer installs callback routines for events and for repaint and resize routines, and an input mask is set for notification of certain events that the application is interested in. However, like normal panels, the scrollable panel does this automatically. Other than this, scrollable panels may take the same attributes as normal panels.

7.2 Creating Panel Items

Like other XView object, panel items are created using `xv_create()`:

```
Panel_item
xv_create(panel, item_type, attrs)
    Panel panel;
    <item type>             item_type;
    <attribute-value list> attrs;
```

The value of `item_type` must be a panel item from one of the panel item packages:

- `PANEL_ABBREV_MENU_BUTTON`
- `PANEL_BUTTON`
- `PANEL_CHECK_BOX`
- `PANEL_CHOICE`
- `PANEL_CHOICE_STACK`
- `PANEL_DROP_TARGET`
- `PANEL_GAUGE`
- `PANEL_LIST`
- `PANEL_MESSAGE`
- `PANEL_MULTILINE_TEXT`
- `PANEL_NUMERIC_TEXT`
- `PANEL_SLIDER`
- `PANEL_TEXT`
- `PANEL_TOGGLE`

The items in this list represent the items found in Figure 7-4. Each item's type can be retrieved by calling:

```
Panel_item_type type;
type = (Panel_item_type)xv_get(panel_item, PANEL_ITEM_CLASS);
```

`Panel_item_type` is an enumerated type found in <*xview/panel.h*> and contains the following types:

- `PANEL_ABBREV_MENU_BUTTON_ITEM`
- `PANEL_BUTTON_ITEM`
- `PANEL_CHOICE_ITEM`
- `PANEL_DROP_TARGET_ITEM`
- `PANEL_GAUGE_ITEM`
- `PANEL_LIST_ITEM`
- `PANEL_MESSAGE_ITEM`
- `PANEL_MULTILINE_TEXT_ITEM`
- `PANEL_NUMERIC_TEXT_ITEM`
- `PANEL_SLIDER_ITEM`
- `PANEL_TEXT_ITEM`
- `PANEL_TOGGLE_ITEM`
- `PANEL_EXTENSION_ITEM`

Most panel items have no windows associated with them (internally, the package uses Xlib calls to draw the items directly onto the panel window). There are no windows associated with panel items, so panel items should be careful not to overlap one another because one item will not *clip* the one under it. For event processing, when an event occurs in a panel, the package scans through items in the panel in the order that they were created. The event is dispatched to the first item whose item rectangle includes the x,y coordinate of the event. If there is another item that shares the same space, then it will not see the event. For this reason, it is important that panel items are *tiled* and do not intersect partially or completely.

There are cases where the panel item displayed in a particular location is dependent on the state of the application. In such cases, only one panel item would be visible and the other(s) would be hidden. If you need to use more than one panel item at the same location, make one panel item visible by setting `XV_SHOW` to `TRUE`, and hide the other panel item by setting `XV_SHOW` to `FALSE`. Depending on the state of the application, you can toggle the `XV_SHOW` values in each of the panel items.

7.3 Layout of Panels and Panel Items

This section covers the layout of panels and panel items. The layout for panels describes their orientation and the spacing between panel items. The section describing the layout of panel items shows the components of a panel item. Normally, applications do *not* need to alter the default layout mechanism for panels or for panel items.

7.3.1 Panel Layout

A panel lays out panel items in rows and columns. The width and height of the rows and columns may be set by the `PANEL` package automatically, or by setting window attributes, as well as by setting the sizes of the panel items themselves. The default layout mechanism is usually all that is necessary. You do not need to give explicit layout methods for the general case. Using this approach, the width and height of panels may change dynamically and you will not need to adjust the positions for panel items. Because absolute positioning methods do not allow for dynamic scaling, they are *not* generally recommended.

Whenever panel items are created, they are added to the adjacent panel item in either a row-first or a column-first order. As each new item is added, it is placed at the next position depending on the value of the panel's `PANEL_LAYOUT` attribute. The default value, `PANEL_HORIZONTAL`, lays out panel items horizontally until the items have reached the edge of the panel. A new row is then started and the next item is placed at the left edge of the panel. Setting the value of `PANEL_LAYOUT` to `PANEL_VERTICAL` causes items to be laid out by column first; when the height of the panel is reached, the next column is started.

If you plan to fit many panel items in a single row or column and want to use default positioning and `window_fit()` (see Section 7.5, "Sizing Panels") it may be necessary to create the panel with a size greater than the default width or height. Otherwise, items could be placed in a new row or column before expected.

The gap between items as they are laid out vertically and horizontally can be set by the attributes `PANEL_ITEM_X_GAP` and `PANEL_ITEM_Y_GAP`. The default gap is 10 pixels in the horizontal direction and 13 pixels in the vertical direction. The panel gap can be set on an entire panel or on individual panel items within a panel.

When you want the next item to start a new row and the panel layout is `PANEL_HORIZONTAL`, use the attribute `PANEL_NEXT_ROW`. This specifies that the item is to start a new row and is to be offset by the number of pixels specified. Setting `PANEL_NEXT_ROW` to the value –1 uses the value of `PANEL_ITEM_Y_GAP` for the row gap.

Similarly, when you want the next item to start a new column, and the panel layout is `PANEL_VERTICAL`, use the attribute `PANEL_NEXT_COL`. It specifies that the item is to start a new column is to be offset by the number of pixels specified. Setting `PANEL_NEXT_COL` to value –1 uses the value of `PANEL_ITEM_X_GAP` column gap (see Section 7.4, "Explicit Panel Item Positioning").

Since different panel items have different sizes, the grid is not rigidly adhered to and the position of items may fluctuate within a row. However, it is possible to force items to line up to specific rows and columns, as will be shown later.

Figure 7-4 shows how panels items are laid out as they are created.

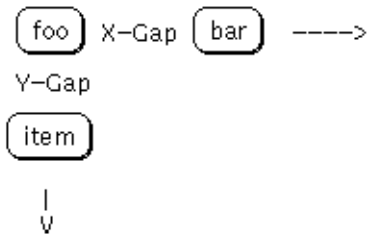

Figure 7-4. Layout of panel items

7.3.2 Panel Item Layout

A panel item is made up of two parts: a label and a value. The area for a panel item's label is called the *label rectangle*. The area for a panel item's value is called the *value rectangle*. The entire area where a panel item may be placed is the panel item's *item rectangle*. The components of a panel item are shown in Figure 7-5. Some panel items, like `PANEL_MESSAGE` and `PANEL_BUTTON`, only have a label. Their value rectangle has a width and height of 0. Other panel items may not have a label; in these cases, the label rectangle has a width and height of 0 (e.g., `PANEL_TEXT`; however, if a `PANEL_TEXT` item does have a label, the label rectangle width and height will be greater than 0).

Except where mandated by OPEN LOOK (e.g., `PANEL_SLIDER`), the panel item attribute `PANEL_LAYOUT` determines the orientation between the label and value. If the layout is horizontal, the value rectangle is placed to the right of the label rectangle. If the layout is vertical, the value rectangle is placed below the label rectangle. Note that `PANEL_LAYOUT` is both a panel and a panel item attribute. However, these represent two different values.

You can embed newlines into strings supplied to the `PANEL_LABEL_STRING` attribute. The newlines tell XView to generate multi-line labels for panel items. Note the sub-strings are right-justified. By default, using multi-line labels, the value rectangle is placed after the last line of the label rectangle.

The attribute `PANEL_BORDER` adds a border around the panel. In a 3D implementation, this border is two pixels wide and presents a "chiseled" appearance. In 2D, the border is one pixel wide. Since the border is rendered directly on the Panel, it is the job of the application to make sure that no Panel_item's overlap the border.

7.4 Explicit Panel Item Positioning

In some cases, you may need to alter the the default panel item positioning. To position panel items explicitly, you can use one of two general methods: relative panel item positioning, or absolute panel item positioning. Using relative panel item positioning, you can adjust your panel items and still accommodate various font sizes. This section describes both of these methods for positioning panel items.

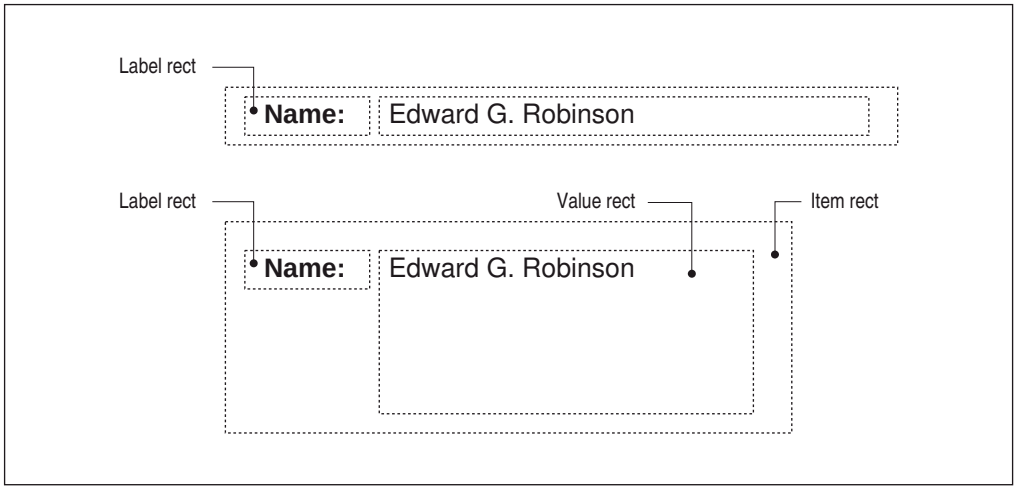

Figure 7-5. Panel item value rectangle and label rectangle

7.4.1 Relative Panel Item Positioning

You can use either of two general methods to accomplish relative panel item positioning. The first method uses the attributes `PANEL_ITEM_X_GAP` and `PANEL_ITEM_Y_GAP` to change the size of the row and column gap between panel items. In the following example, these attributes are used to change the gap between columns to 20 pixels and the gap between rows to 10 pixels:

```
xv_set(panel,
       PANEL_ITEM_X_GAP, 20,
       PANEL_ITEM_Y_GAP, 10,
       NULL);
/* Create all the panel items */
xv_create(panel, PANEL_BUTTON,
       PANEL_LABEL_STRING, "First",
       PANEL_NOTIFY_PROC, first_notify_proc,
       NULL);
xv_create(panel, PANEL_BUTTON,
       PANEL_LABEL_STRING, "Second",
       PANEL_NOTIFY_PROC, second_notify_proc,
       NULL);
```

In the second method for relative item positioning, each panel item you create is laid out relative to the previously created item. After an item is created, you get its XV_RECT and add individual gap values. You then use XV_X and XV_Y to position the next panel item. The following example demonstrates this method:

```
first = xv_create(panel, PANEL_BUTTON,
       PANEL_LABEL_STRING, "First",
       PANEL_NOTIFY_PROC, first_notify_proc,
       NULL);
rect = (Rect *) xv_get(first, XV_RECT);
     /* Position the next item 20 pixels to the right*/
     /* right of the previous item                   */
second = xv_create(panel, PANEL_BUTTON,
       PANEL_LABEL_STRING, "Second",
       PANEL_NOTIFY_PROC, second_notify_proc,
       XV_X, rect_right(rect) + 20,
       NULL);
rect = (Rect *) xv_get(second, XV_RECT);
     /* Position the next item 30 pixels to the right */
     /* of the previous item                          */
third = xv_create(panel, PANEL_BUTTON,
       PANEL_LABEL_STRING, "Third",
       PANEL_NOTIFY_PROC, third_notify_proc,
       XV_X, rect_right(rect) + 30,
       NULL);
```

7.4.2 Absolute Panel Item Positioning

Using absolute panel item positioning, the attributes XV_X and XV_Y specify the position of panel items explicitly. These attributes specify absolute positioning relative to the panel. The attributes PANEL_ITEM_X and PANEL_ITEM_Y are specific to panels. These values reflect the coordinates of the last item created. Therefore, they are both get-only attributes. The following code fragment shows how both are used:

```
panel = (Panel)xv_create(frame, PANEL, NULL);

xv_create(panel, PANEL_BUTTON,
    XV_X,               50,
    XV_Y,               75,
    PANEL_LABEL_STRING, "Quit",
    NULL);

printf("The last item in the panel was at %d %d.0,
    xv_get(panel, PANEL_ITEM_X), xv_get(panel, PANEL_ITEM_Y));
```

This code segment would print:

```
The last item in the panel was at 50 75.
```

When positioning panel items, if the position of the new panel item extends beyond the edges of the panel in the positive direction, the size of the panel increases to include the item. This happens regardless of whether the panel's size was set explicitly at the time it was created.*

*Setting an item to negative coordinates does not increase the size of the panel—only if the item is wide enough or high enough to stretch into the panel window will any of it be visible.

7.4.2.1 General positioning of items

Two functions that are available for general positioning of panel items within windows are `xv_row()` and `xv_col()`. These functions use the values of `WIN_ROW_GAP` and `WIN_COLUMN_GAP` of the panel. While these attributes control the spacing between panel items, the distance between items and the edge of the panel is set to a constant 4 pixels.

Consider the following code fragment that positions items within regimented rows and columns:

```
int         rows, cols;
extern char *names[3][5];

frame = (Frame)xv_create(NULL, FRAME, NULL);
panel = (Panel)xv_create(frame, PANEL,
    WIN_ROW_GAP,          70,
    WIN_COLUMN_GAP,       20,
    NULL);

for (rows = 0; rows < 3; rows++)
    for (cols = 0; cols < 5; cols++)
        (void) xv_create(panel, PANEL_BUTTON,
            XV_X,               xv_col(panel, cols),
            XV_Y,               xv_row(panel, rows),
            PANEL_LABEL_STRING, names[rows][cols],
            PANEL_NOTIFY_PROC,  (rows+cols==0)? quit : selected,
            PANEL_CLIENT_DATA,  frame,
            NULL);
```

This code displays a panel with a 70-pixel distance between the upper-left corners of the panel items. Thus, each panel item must be less than 70 pixels wide or they will overlap one another. This contrasts with `PANEL_ITEM_X_GAP` and `PANEL_ITEM_Y_GAP`, which specify the gap between the right and left sides of adjacent panel items. Note that this does not affect the distance between the edge of the panel and the items along the perimeter; that distance remains constant at 4 pixels.

The `xv_col()` and `xv_row()` method of explicit panel item placement does not take into account the sizes of the panel items. If not enough space is given between the rows and columns (`WIN_COLUMN_GAP`, `WIN_ROW_GAP`), then items will lie on top of one another. Neither does this method take into account dynamic scaling of items and fonts. As a result, this and all absolute positioning methods are *not* recommended.

7.4.3 Layout of Panel Items with Values

For panel items with values, `PANEL_VALUE_X` and `PANEL_VALUE_Y` can be used instead of the `XV_X` and `XV_Y` attributes mentioned above to align the value rectangles of panel items. The label portion of the panel item is then positioned automatically according to the panel layout characteristics currently in effect. This is useful, for example, when you need to align a series of `PANEL_TEXT` items. It is important *not* to use `XV_X` and `XV_Y` when using the label and value positioning attributes.

7.5 Sizing Panels

The size of a panel, by default, extends to the bottom and right edges of the frame in which it is placed (assuming there are no other subwindows in the frame). Alternatively, the panel's dimensions can be set explicitly using `XV_WIDTH` and `XV_HEIGHT`. If it is important to maintain the layout of the panel items in a panel, then the dimensions should be set explicitly.

More often than not, you want the panel to be just the minimum height and width required to encompass all of its items. You can set the minimum height and width using the macros `window_fit_height()` or `window_fit_width()`, respectively. You can set both in a single call to `window_fit()`. These macros are called after all the items have been created.

The attributes `PANEL_EXTRA_PAINT_WIDTH` and `PANEL_EXTRA_PAINT_HEIGHT` specify the increment by which a panel will grow in the *x* and *y* directions, respectively.

7.6 Panel Item Values

Many panel items are associated with a specific value. A text item has a *string* value, a numeric text item has an integer value, a choice item has a current-choice value, and so on.* To set a value, use:

```
xv_set(item, PANEL_VALUE, value, NULL)
```

Of course, the type of `value` is dependent on the type of panel item whose value is being set. Consider the following examples:

```
/* Set the text field in the text item to print "Hello World" */
xv_set(text_item, PANEL_VALUE, "Hello world.", NULL);

/* Set the numeric value of the numeric text item to be 10 */
xv_set(text_num_item, PANEL_VALUE, 10, NULL);

/* Set the current choice in choice_item to be the fifth choice */
xv_set(choice_item, PANEL_VALUE, 4, NULL);
```

NOTE

The values for string-valued attributes are dynamically allocated when they are created or set. The value you specify is *copied* into the newly allocated space. If a previous value was present, the panel item frees its old data first.†

Panel item values are retrieved in a similar way:

```
xv_get(item, PANEL_VALUE);
```

---

*For details about the value type of a panel item, see the corresponding sections on specific panel items (Sections 7.9 through 7.18).

†This contrasts with menu items. See Chapter 11, *Menus*.

Since the xv_get() routines are used to retrieve attributes of all types, you should cast the value returned into the type appropriate to the attribute being retrieved:

```
int val;
val = (int)xv_get(num_text_item, PANEL_VALUE);
printf("The int-value in the num_text_item is: '%d'0, val);
```

NOTE

xv_get() does not dynamically allocate storage for the values it returns. If the value returned is a pointer, it points directly into the panel's private data. It should be considered *read-only*—do not change the contents of the pointer; it is your responsibility to copy the information pointed to.

7.7 Iterating Over a Panel's Items

You can iterate over each item in a panel with the two attributes PANEL_FIRST_ITEM and PANEL_NEXT_ITEM. A pair of macros, PANEL_EACH_ITEM() and PANEL_END_EACH are also provided for this purpose. For example, to destroy each item in a panel:

```
Panel_item item;

PANEL_EACH_ITEM(browser, item)
    xv_destroy(item);
PANEL_END_EACH
```

Note that a semicolon is not required after PANEL_END_EACH.

7.8 Panel Item Classes

Nine types of panel items are presented here:

- Panel Buttons, Menu Buttons, and Abbreviated Menu Buttons
- Checkboxes
- Exclusive and Nonexclusive Choices
- Abbreviated Choices
- Scrolling Lists
- Message Items
- Sliders
- Text Items (including numeric and multiline text items)
- Drop Target Items

Items are made up of one or more displayable components. One component shared by all item types is the *label*. An item label is either a string or a graphic image.

The user interacts with items through various methods ranging from mouse button selection to keyboard input. This interaction typically results in a *callback* function being called for the panel item. The callback functions also vary on a per-item basis. Each item type is described in the following sections.

7.9 Button Items

A button item allows the user to invoke a command or bring up a menu. Examples of various buttons are listed in Figure 7-6. The button's label identifies the name of the command or menu. A button label that ends in three dots (. . .) indicates that a pop-up menu will be displayed when the button is selected.

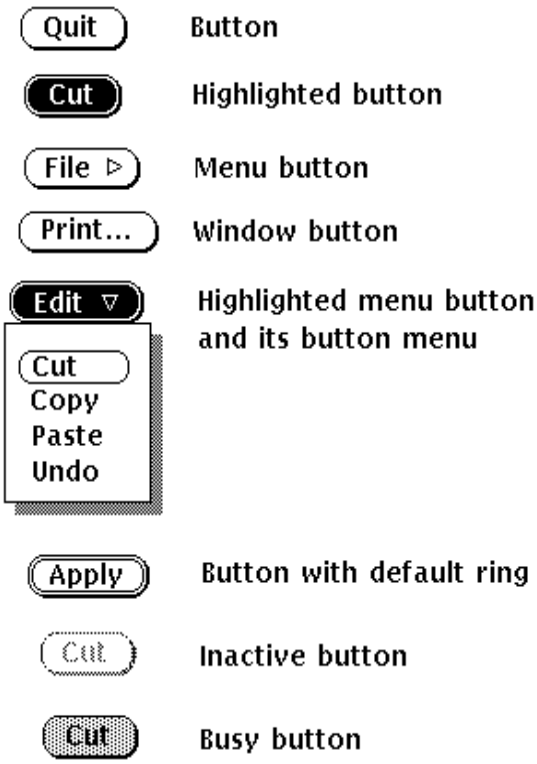

Figure 7-6. Visual feedback for button controls

A button requires a label specified by the attribute `PANEL_LABEL_STRING` or `PANEL_LABEL_IMAGE`. Buttons do not have a `PANEL_VALUE` associated with them.

7.9.1 Button Selection

The user invokes a panel button by clicking the SELECT mouse button on it. When this happens, the button's notify procedure is called. The notify procedure is installed by specifying PANEL_NOTIFY_PROC, as in the following example:

```
xv_create(panel, PANEL_BUTTON,
    PANEL_NOTIFY_PROC,   quit_proc,
    PANEL_LABEL_STRING,  "Quit",
    NULL);
```

When the button is selected, the notify procedure is called. The form of the notify procedure for a button is:

```
void
button_notify_proc(item, event)
    Panel_item  item;
    Event      *event
```

The function does not return a value, but if the action that the button had intended to take fails, then you should set the item's PANEL_NOTIFY_STATUS to XV_ERROR (e.g., if the button was labeled "save" but the actual save operation failed). It is set to XV_OK by default. If the button is part of an unpinned Command Frame, setting PANEL_NOTIFY_STATUS to XV_ERROR will prevent the Command Frame from being dismissed. Use MENU_NOTIFY_STATUS for a menu. In a callback, setting this to XV_ERROR on the menu that was notified prevents the frame from being dismissed.

When a button's notify procedure is called, the button's *busy* state is set. When a button's busy state is set to TRUE, the button does not accept further input (clicking the SELECT mouse button will do nothing to the button item). The busy state is cleared when the notify procedure exits. The busy state can be maintained after exiting the notify procedure by setting PANEL_BUSY to TRUE from within the notify procedure.

7.9.1.1 Making a button inactive

The attributes PANEL_INACTIVE is used to make a panel button inactive. If TRUE, the button item cannot be selected. Inactive items are displayed with gray-out pattern as shown in Figure 7-6.

7.9.2 Menu Buttons

It is often useful to attach a menu to a button. The menu may provide alternate values or functions for the button to invoke. Since the menu is a separate entity (in other words, it is not created when the button is created—you have to create it on your own), the menu may have callback routines associated with it and its menu items.*

*Read Chapter 11, *Menus*, for details of menus and menu items.

When a menu button receives an ACTION_MENU down event, the button's notify procedure is called before the menu is displayed. This gives you the chance to modify the menu beforehand. When the menu button is selected by clicking the SELECT button, the button's notify procedure is called before the menu's notify procedure.

Menu buttons contain a triangle pointing in the direction in which the menu will be displayed.

The *btn_menu.c* program in Example 7-2 demonstrates how a menu can be attached to a panel button.

Example 7-2. The btn_menu.c program

```
/*
 * btn_menu.c -- display a panel that has an OPEN LOOK menu button.
 * The choices displayed are Yes, No and Quit.  If Quit is selected
 * in the menu, the program exits.
 */
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/openmenu.h>

main(argc, argv)
int argc;
char *argv[ ];
{
    Frame       frame;
    Panel       panel;
    Menu        menu;
    int         selected();
    void        menu_proc();

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

    frame = (Frame)xv_create(NULL, FRAME, NULL);
    panel = (Panel)xv_create(frame, PANEL, NULL);

    /* Create the menu _before_ the panel button */
    menu = (Menu)xv_create(NULL, MENU,
        MENU_NOTIFY_PROC,       menu_proc,
        MENU_STRINGS,           "Yes", "No", "Quit", NULL,
        NULL);
    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Y/N/Q",
        PANEL_NOTIFY_PROC,      selected,
        PANEL_ITEM_MENU,        menu, /* attach menu to button */
        NULL);
    window_fit(panel);
    window_fit(frame);
    xv_main_loop(frame);
}

int
selected(item, event)
Panel_item item;
Event *event;
{
```

Example 7-2. The btn_menu.c program (continued)

```
    printf("%s selected...\n", xv_get(item, PANEL_LABEL_STRING));
    return XV_OK;
}

void
menu_proc(menu, menu_item)
Menu menu;
Menu_item menu_item;
{
    printf("Menu Item: %s\n", xv_get(menu_item, MENU_STRING));
    if (!strcmp((char *)xv_get(menu_item, MENU_STRING), "Quit"))
        exit(0);
}
```

The output produced by this program is shown in Figure 7-7.

7.9.2.1 Destroying menu buttons

XView automatically destroys the menu associated with the button unless there are other references to the menu. For each button or any other XView object that references the menu in question, the menu's `XV_REF_COUNT` is incremented. Thus, if five buttons reference the same menu, then destroying one button will not destroy the menu, but it will decrement the reference count by one. If the destruction of the panel item would decrement the menu's reference count to `0`, then the menu itself is destroyed. Therefore, if you wish to prevent the menu from being destroyed, you can forcefully increment the menu's reference count by at least one more than what it is. That way, no matter how many times the menu is used, its reference count will always be set to a value greater than zero, and it will never be destroyed by destroying panel items that use it. Of course, you can always call `xv_destroy()` on the menu explicitly when you want to destroy the menu once and for all.

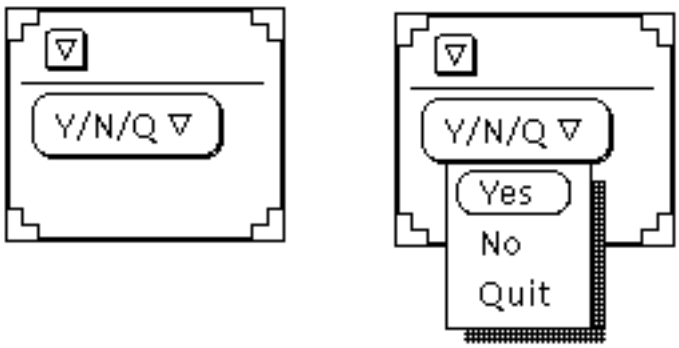

Figure 7-7. Sample menu button (unselected and selected)

You can use the attributes `XV_INCREMENT_REF_COUNT` and `XV_DECREMENT_REF_COUNT` on any item.

```
xv_set(menu, XV_INCREMENT_REF_COUNT, NULL);
```

This is not recommended except when you need to circumvent the normal functionality of XView.

7.9.3 Panel Button Width

The width of a panel button's string or image is available and can be set with the attribute PANEL_LABEL_WIDTH. This attribute does not include the width of a panel button's end caps or menu marks. PANEL_LABEL_WIDTH has no effect on a PANEL_BUTTON until the panel button's PANEL_LABEL_STRING or PANEL_LABEL_IMAGE is set.

To make all the panel buttons the same width for a group of buttons that are not all menu buttons or all non-menu buttons, requires three steps:

1. Set the PANEL_LABEL_WIDTH for each button.
2. Find the maximum XV_WIDTH of all the buttons in the group. This value includes the width of any end caps or menu marks.
3. Finally, add the difference between each button's XV_WIDTH and the maximum XV_WIDTH of all the buttons to each button's PANEL_LABEL_WIDTH:

```
xv_set(item, PANEL_LABEL_WIDTH,
       (int)xv_get(item, PANEL_LABEL_WIDTH) +
       max_XV_WIDTH - (int)xv_get(item,XV_WIDTH),
       NULL);
```

7.9.4 Abbreviated Menu Buttons

Abbreviated menu buttons are just like menu buttons. However, they do not display the label inside the button, but to the left, as shown in Figure 7-8.

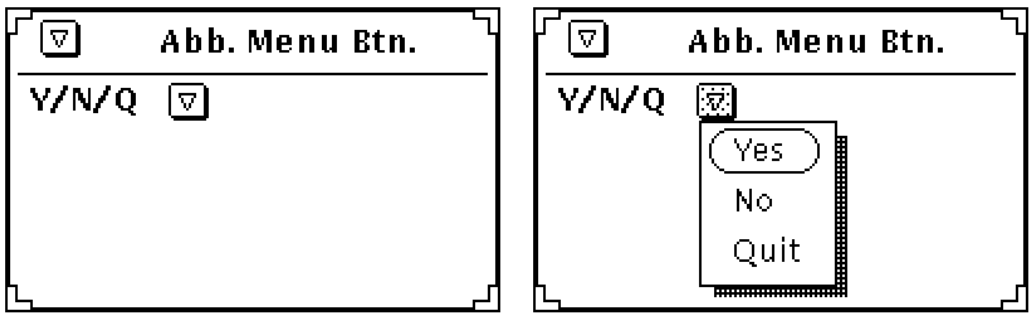

Figure 7-8. Sample abbreviated menu button

Abbreviated menu items are created using the PANEL_ABBREV_MENU_BUTTON package.

```
Panel_item item;
extern Menu menu; /* created separately */

item = xv_create(panel, PANEL_ABBREV_MENU_BUTTON,
    PANEL_ITEM_MENU,   menu,
    NULL);
```

Notification is the same as in full-size menu buttons.

7.10 Choice Items

Choice items provide a list of different choices to the user in which one or more choices may be selected. There are variations of choice items which implement different OPEN LOOK objects such as:

- Exclusive and Nonexclusive Choices (or Settings)
- Abbreviated Choice Items
- Checkboxes

Behind the flexibility of presentation lies a uniform structure consisting of a label, a list of choices and an indication of which choice or set of choices is currently selected. The choices can be displayed as either text strings (PANEL_CHOICE_STRINGS) or server images (PANEL_CHOICE_IMAGES). The number of choices in a choice list is returned by the PANEL_NCHOICES attribute. This is true for toggle items as well.

7.10.1 Display and Layout of Item Choices

The attribute PANEL_DISPLAY_LEVEL determines which of an item's choices are actually displayed. The display level may be set to:

PANEL_ALL	The default. All choices are shown.
PANEL_CURRENT	Only the current choice is shown.
PANEL_NONE	No choices are shown.

The choices are laid out either horizontally or vertically next to the label depending on the value of the item's value for PANEL_LAYOUT. By default, this value is PANEL_HORIZONTAL.

Sometimes the number of choices in a choice list can get long and the menu for the item (if any) may look aesthetically bad or not fit on the screen. You can specify that choices appear in row and column layout by specifying either the number of rows or the number of columns with the attributes PANEL_CHOICE_NROWS and PANEL_CHOICE_NCOLS. If both are specified, the last one specified takes precedence.

7.10.2 Exclusive and Nonexclusive Choices

When a default choice item is created, its type is an exclusive choice item allowing the user to select only one choice from the list. The value of the panel item is the currently selected choice. The index of the first choice is 0. In the following example, we create several choice items as exclusive settings:

```
xv_create(panel, PANEL_CHOICE,
    PANEL_LABEL_STRING,     "Choice - Exclusive",
    PANEL_CHOICE_STRINGS,   "One", "Two", "Three", "Four", NULL,
    PANEL_NOTIFY_PROC,      selected,
    PANEL_VALUE,            3,
    NULL);
```

This code fragment produces the panel item shown in Figure 7-9.

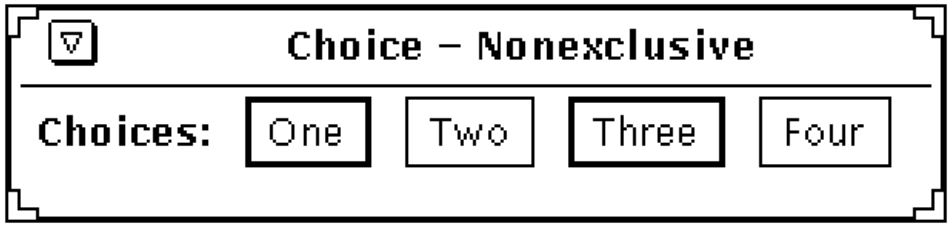

Figure 7-9. Sample panel with exclusive choices

Figure 7-9 represents a panel item that has four choices, the fourth of which is *set*. If the user makes another choice, the value of the item, and therefore the value of `PANEL_VALUE`, changes to the ordinal number of the choice.

The choice item can be made nonexclusive, allowing more than one of the choices to be selected, when the attribute `PANEL_CHOOSE_ONE` is set to `FALSE`. The macro `PANEL_TOGGLE` has been defined as:

```
PANEL_CHOICE, PANEL_CHOOSE_ONE, FALSE
```

This macro affects the panel item in two ways. More than one choice may be set in the visual feedback, and the value of the item is set as a mask indicating the choices selected. In the following example, the choice items in the previous example are shown as nonexclusive settings:

```
xv_create(panel, PANEL_TOGGLE,
    PANEL_LABEL_STRING,     "Choice - Nonexclusive",
    PANEL_CHOICE_STRINGS,   "One", "Two", "Three", "Four", NULL,
    PANEL_NOTIFY_PROC,      selected,
    PANEL_VALUE,            5,
    NULL);
```

This code fragment would produce the panel item shown in Figure 7-10.

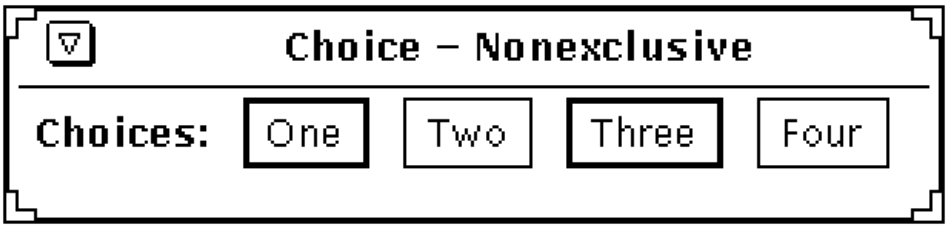

Figure 7-10. Sample panel with nonexclusive choices

Figure 7-10 represents a panel item that has two choices set, the first and the third. The reason for this is that the panel's value is set to 5, which is a mask that represents the first and third bits. For example, 5 in binary is ... 00101.* In the binary representation, the first and third bits from the right are 1's. This means that the first and third choices are selected. This is how the value is interpreted on calls to `xv_set()` or `xv_create()`, and how it is returned for calls to `xv_get()`.

To get the image for a choice item's current choice (assuming the choice item is a `Server_image` choice):

```
Server_image image;
image = (Server_image)xv_get(item, PANEL_CHOICE_IMAGE,
                             xv_get(item, PANEL_VALUE));
```

For choice items whose `PANEL_CHOOSE_NONE` value is `TRUE`, a `PANEL_VALUE` of -1 may be set or returned, indicating that no choices are set for that item.

Setting `PANEL_CHOOSE_NONE` allows choice items to have no currently selected item. This attribute is not applicable if `PANEL_CHOOSE_ONE` is `FALSE`.

7.10.3 Abbreviated Choices

Abbreviated choices are exclusive choices that either display no value or only the current value. A menu is used to display all the choices. To implement abbreviated choice items, the macro `PANEL_CHOICE_STACK` is used. This macro creates an abbreviated choice item that displays only the current value. It is defined as:

```
PANEL_CHOICE, PANEL_DISPLAY_LEVEL, PANEL_CURRENT
```

To create an abbreviated choice item that does *not* display the current value, use the `PANEL_ABBREV_MENU_BUTTON` package and set `PANEL_DISPLAY_LEVEL` to `PANEL_NONE` (refer to Section 7.9.4, "Abbreviated Menu Buttons," for details). The following example demonstrates creating an abbreviated choice item that displays the current value:

```
xv_create(panel, PANEL_CHOICE_STACK,
    PANEL_LAYOUT,           PANEL_VERTICAL,
```

*The value for nonexclusive choice items is stored as an `unsigned int` and the maximum number of nonexclusive choice items is 32.

```
    PANEL_LABEL_STRING,     "Abbreviated Choice",
    PANEL_CHOICE_STRINGS,   "One", "Two", "Three", "Four", NULL,
    PANEL_NOTIFY_PROC,      selected,
    PANEL_VALUE,            1,
    NULL);
```

The panel item created by this code is shown in Figure 7-11.

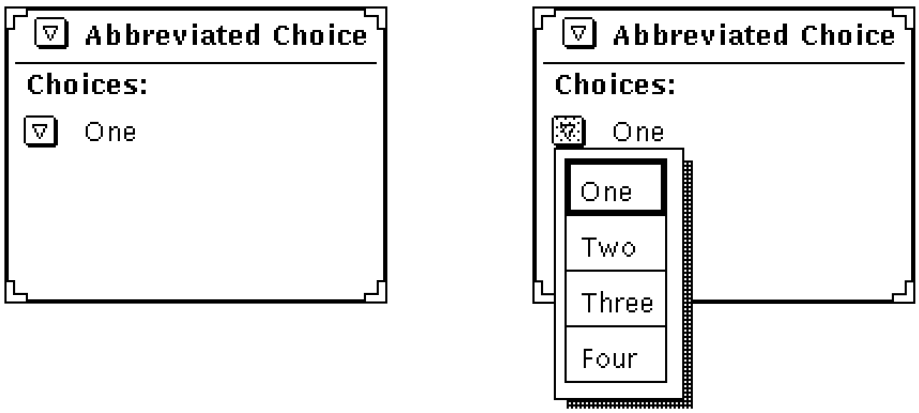

Figure 7-11. Sample panel with abbreviated choice (unselected and selected)

Here, since only the current selection is visible, the only way to make other choices in the item is to bring up a menu. The value of the panel item is the same as for an exclusive `PANEL_CHOICE` item.

7.10.4 Checkbox Choices

Checkboxes are nonexclusive choices that use checkmarks to indicate the selected choices. Unselected choices have empty checkboxes. The following example demonstrates checkboxes:

```
xv_create(panel, PANEL_CHECK_BOX,
    PANEL_LAYOUT,           PANEL_HORIZONTAL,
    PANEL_LABEL_STRING,     "Choices",
    PANEL_CHOICE_STRINGS,   "One", "Two", "Three", "Four", NULL,
    PANEL_NOTIFY_PROC,      selected,
    PANEL_VALUE,            5,
    NULL);
```

The panel item created by this code is shown in Figure 7-12.

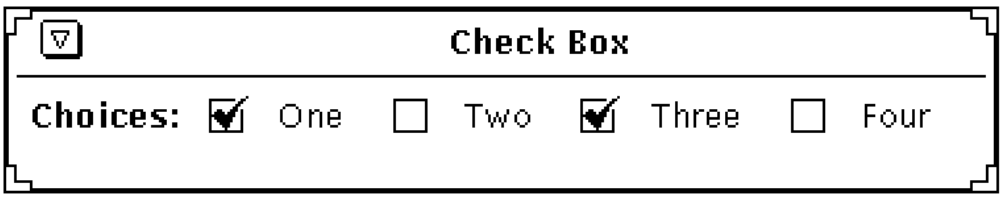

Figure 7-12. Sample panel with checkbox

All of the choices can be selected in the same way as a PANEL_TOGGLE.

7.10.5 Choice Selection and Notification

The user can make a selection from a choice item by selecting the desired choice directly with the SELECT mouse button.

The procedure specified via the attribute PANEL_NOTIFY_PROC will be called when any of its choices are selected. If a choice item's current selection or value changes as a result of a call to xv_set() from somewhere else, then the notify procedure is not called. The choice notify procedure is passed the item, the current value of the item, and the event that caused notification:

```
void
choice_notify_proc(item, value, event)
    Panel_item  item;
    int         value;
    Event      *event;
```

Like the button's notify procedure, the choice notify procedure is also a void function. If the function fails to perform its task, you should set the item's PANEL_NOTIFY_STATUS to XV_ERROR.

For exclusive choices, the value passed to the notify procedure is the ordinal number corresponding to the current choice (the choice that the user has just selected). The first choice has ordinal number zero. For nonexclusive choices, the value is a mask of the currently selected choices in the list (see Section 7.11.3, "List Selection").

The event is the event that caused the notify procedure to be called. For these types of choices, the event action will probably be ACTION_SELECT.

7.10.6 Foreground Color in Choice Items

Colors for panel choice items may be set with the PANEL_CHOICE_COLOR attribute. This attribute sets the foreground color index for specified choices. The following example demonstrates a choice using the foreground color attribute.

```
xv_create(panel, PANEL_TOGGLE,
    PANEL_LABEL_STRING,     "Choices",
    PANEL_CHOICE_STRINGS,   "One", "Two", "Three", "Four", NULL,
    PANEL_NOTIFY_PROC,      selected,
    PANEL_VALUE,            5,
    PANEL_CHOICE_COLOR, 0, RED, NULL,
    PANEL_CHOICE_COLOR, 1, BLUE, NULL,
    PANEL_CHOICE_COLOR, 2, RED, NULL,
    PANEL_CHOICE_COLOR, 3, BLUE, NULL,
    NULL);
```

7.10.7 Parallel Lists

Parallel lists are lists of values for particular attributes that correspond to each choice in the panel item. An example of a parallel list is `PANEL_CHOICE_XS` and `PANEL_CHOICE_YS`. These two attributes take as values a `NULL`-terminated list of coordinates to specify explicit placement of the choices when they are displayed (assuming `PANEL_ALL` is the display format).

WARNING

> The attributes `PANEL_CHOICE_XS`, `PANEL_CHOICE_YS`, `PANEL_CHOICE_X` and `PANEL_CHOICE_Y` are provided for SunView1 Compatibility. They are mentioned here for explanatory purposes only. Their use allows you to create applications that may *not* be OPEN LOOK-compliant.

These attributes are used to display choices in adjacent rows and columns, as in the following example:

```
xv_create(panel, PANEL_CHOICE,
    PANEL_CHOICE_STRINGS,    "One", "Two", "Three", NULL,
    PANEL_CHOICE_XS,         10, 70, 130, NULL,
    PANEL_CHOICE_YS,         90, NULL,
    PANEL_VALUE,             2,
    PANEL_NOTIFY_PROC,       notify_proc,
    NULL);
```

The choice item has three choices: the strings “One”, “Two,” and “Three.” We have specified explicit positioning of the choice items using the attributes `PANEL_CHOICE_XS` and `PANEL_CHOICE_YS`. These attributes take precedence over `PANEL_LAYOUT`, so that layout is ignored if specified. Note that the list `PANEL_CHOICE_YS` has only one element. When any of the parallel lists are abbreviated in this way, the last element given will be used for the remainder of the choices. So, in the example above:

```
90, NULL,
```

serves as shorthand for:

```
90, 90, 90, NULL,
```

All the choices will appear at *y* coordinate 90, while the *x* coordinates for the choices will be 10, 70, and 130, respectively.

You cannot specify that a choice appear at *x = 0* or *y = 0* by using the attributes PANEL_CHOICE_XS or PANEL_CHOICE_YS. Since these attributes take NULL-terminated lists as values, the zero would be interpreted as the terminator for the list. You may achieve the desired effect by setting the positions individually. The attributes PANEL_CHOICE_X or PANEL_CHOICE_Y take as values the number of the choice followed by the desired position. The following example demonstrates setting the position of choice items:

```
Panel_item   choice;
int          i;
extern char *strings[ ];

choice = (Panel_item)xv_create(panel, PANEL_CHOICE,
    PANEL_CHOOSE_ONE,        FALSE,
    NULL);
for (i = 0; i < sizeof(strings) / sizeof(char *); i++)
    xv_set(choice,
        PANEL_CHOICE_STRING, i, strings[i],
        PANEL_CHOICE_X,      i, i*20,
        PANEL_CHOICE_Y,      i, i*20,
        NULL);
```

After the choice item is created, the *x* and *y* positions of the choices are set individually in a loop.

Once a set of choice items is created, the rectangle that encloses a specified choice may be returned using the attribute PANEL_CHOICE_RECT. It takes an integer argument representing the index of the choice whose `rect` pointer is returned.

7.11 Scrolling Lists

OPEN LOOK's specification for *scrolling lists* is implemented by the PANEL_LIST panel item. List items allow the user to make selections from a scrolling list of choices larger than can be displayed on the panel at one time. The selections can be exclusive or nonexclusive, like the choice items outlined in the previous section. The list is made up of strings or images and a scrollbar that functions like any scrollbar in XView, except that it cannot be split.* List items are laid out in rows only—one list entry per row. Below is a code fragment for creating a simple list:

```
xv_create(panel, PANEL_LIST,
    PANEL_LIST_STRINGS,     "One", "Two", "Three", "Four", NULL,
    NULL);
```

The list items produced by this code are shown in Figure 7-13.

*See Chapter 10, *Scrollbars*, for a further description of how scrollbars work.

Figure 7-13. Sample panel with scrolling list

7.11.1 Displaying List Items

You can use either text strings or server images to display the choice to the user; you can even intermix them. You specify the choices either one at a time or in a group. To set only one choice, use `PANEL_LIST_STRING` or `PANEL_LIST_GLYPH`. When creating a new string or glyph entry, if the index into the list specified is larger than the total number of entries, then the new item is added to the end of the list. Use `PANEL_LIST_STRINGS` or `PANEL_LIST_GLYPHS` to set all the choices in a group. If no items exist in the list, the appropriate number of rows are created to fit all of the items. If the list already contains items, then the first *n* rows of items are replaced by the newly specified strings or glyphs (where *n* is the number of strings or glyphs specified).

The width of the list item can be set to explicit pixel values using `PANEL_LIST_WIDTH`. The minimum value for this attribute is 25. This reserves enough space for the list's borders and margins. Setting `PANEL_LIST_WIDTH` to -1 extends the width of the scrolling list box to the edge of the panel. Setting `PANEL_LIST_WIDTH` to 0 sets the width to that of the widest row in the scrolling list. Alternatively, the number of rows that are displayed in the list item can be controlled through the value of the `PANEL_LIST_DISPLAY_ROWS` attribute. This value governs the height, in rows, of the list item.

The default panel font for a scrolling list is the default font for the panel. To specify a particular font for a scrolling list row, use `PANEL_LIST_FONT`, which takes two arguments, a row number and a font. To set the fonts for multiple rows, use `PANEL_LIST_FONTS`. Note that the font specification using either `PANEL_LIST_FONT` or `PANEL_LIST_FONTS` should follow the creation of the rows (for example, by `PANEL_LIST_STRINGS`).

By default, a scrolling list does not have a title. To add a title to a scrolling list, use `PANEL_LIST_TITLE` as in the following code segment:

```
xv_set(panel_list_item, PANEL_LIST_TITLE, "Patterns", NULL);
```

The title appears above the list items. The package makes a copy of the string passed to the `PANEL_LIST_TITLE` attribute. The package also will free the string when the title string is no longer needed.

The height of each row in the list may be set using `PANEL_LIST_ROW_HEIGHT`. All rows have the same height. If the items in the list are glyphs, then the height of each row must be specified by at least the height of the tallest glyph in the list. This should be determined before the list of glyphs is set in the list item. Entries in the list can be either glyphs or strings;

an entry containing both a string and a glyph will display both. The glyph will be on the left and the string will be on the right. Consider the program in Example 7-3.

Example 7-3. The list_glyphs.c program

```
/*
 * list.c -- show a scrolling list with three items in it.
 * Each item is an icon (a pattern) and a string.
 */
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/svrimage.h>

#define gray1_width 16
#define gray1_height 16
static char gray1_bits[ ] = {
    0x55, 0x55, 0xaa, 0xaa, 0x55, 0x55, 0xaa, 0xaa, 0x55, 0x55,
    0xaa, 0xaa, 0x55, 0x55, 0xaa, 0xaa, 0x55, 0x55, 0xaa, 0xaa,
    0x55, 0x55, 0xaa, 0xaa, 0x55, 0x55, 0xaa, 0xaa, 0x55, 0x55,
    0xaa, 0xaa
};

#define gray2_width 16
#define gray2_height 16
static char gray2_bits[ ] = {
    0x11, 0x11, 0x00, 0x00, 0x44, 0x44, 0x00, 0x00, 0x11, 0x11,
    0x00, 0x00, 0x44, 0x44, 0x00, 0x00, 0x11, 0x11, 0x00, 0x00,
    0x44, 0x44, 0x00, 0x00, 0x11, 0x11, 0x00, 0x00, 0x44, 0x44,
    0x00, 0x00
};

#define gray3_width 16
#define gray3_height 16
static char gray3_bits[ ] = {
    0x22, 0x22, 0xee, 0xee, 0x33, 0x33, 0xee, 0xee, 0x22, 0x22,
    0xee, 0xee, 0x33, 0x33, 0xee, 0xee, 0x22, 0x22, 0xee, 0xee,
    0x33, 0x33, 0xee, 0xee, 0x22, 0x22, 0xee, 0xee, 0x33, 0x33,
    0xee, 0xee
};

main(argc, argv)
int     argc;
char    *argv[ ];
{
    Frame       frame;
    Panel       panel;
    Server_image gray1, gray2, gray3;
    extern void exit(), which_glyph();

    xv_init(XV_INIT_ARGS, argc, argv, NULL);

    gray1 = (Server_image)xv_create(NULL, SERVER_IMAGE,
        XV_WIDTH,               gray1_width,
        XV_HEIGHT,              gray1_height,
        SERVER_IMAGE_BITS,      gray1_bits,
        NULL);
    gray2 = (Server_image)xv_create(NULL, SERVER_IMAGE,
```

Example 7-3. The list_glyphs.c program (continued)

```
        XV_WIDTH,               gray2_width,
        XV_HEIGHT,              gray2_height,
        SERVER_IMAGE_BITS,      gray2_bits,
        NULL);
    gray3 = (Server_image)xv_create(NULL, SERVER_IMAGE,
        XV_WIDTH,               gray3_width,
        XV_HEIGHT,              gray3_height,
        SERVER_IMAGE_BITS,      gray3_bits,
        NULL);
    frame = (Frame)xv_create(NULL, FRAME, NULL);
    panel = (Panel)xv_create(frame, PANEL, NULL);

    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "quit",
        PANEL_NOTIFY_PROC,      exit,
        NULL);

    (void) xv_create(panel, PANEL_LIST,
        PANEL_LIST_ROW_HEIGHT,   16,
        PANEL_LIST_GLYPHS,       gray1, gray2, gray3, NULL,
        PANEL_LIST_STRINGS,      "Pattern1", "Pattern2", "Pattern3", NULL,
        PANEL_LIST_CLIENT_DATAS, 1, 2, 3, NULL,
        PANEL_NOTIFY_PROC,       which_glyph,
        NULL);

    window_fit(panel);
    window_fit(frame);

    xv_main_loop(frame);
}

void
which_glyph(item, string, client_data, op, event, row)
Panel_item     item; /* panel list item */
char          *string;
caddr_t        client_data;
Panel_list_op  op;
Event         *event;
int            row;
{
    printf("item = %s (#%d), op = %d0, string, client_data, op);
}
```

The output produced by *list_glyphs.c* is shown in Figure 7-14.

The height of each row in the list is determined by the height of the scrolling list font. If glyphs are used, the programmer is responsible for row height. In the example, all the glyphs are the same height (16 pixels), so the calculation is easy: `PANEL_LIST_ROW_HEIGHT` is set to 16. If a glyph exceeds the row height, then a warning is printed and the glyph is ignored.

The use of the notify procedure is discussed in Section 7.11.4, “List Notification.”

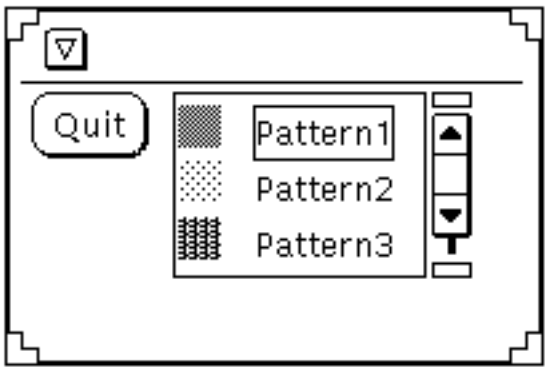

Figure 7-14. Output of program list_glyphs.c

7.11.2 Adding and Deleting List Entries

List entries are denoted by row number. The first entry is row 0. Entries are added and deleted from the list at run time. Several attributes enable you to add and delete entries in a list item. The attributes that let you delete list entries are: `PANEL_LIST_DELETE`, `PANEL_LIST_DELETE_ROWS`, and `PANEL_LIST_DELETE_SELECTED_ROWS`. Using `PANEL_LIST_DELETE`, the attribute value specifies a single list item to delete. The string and/or image resources are deallocated and the list is updated appropriately.*

The attribute `PANEL_LIST_DELETE_ROWS` deletes multiple list item rows. This attribute takes two integer arguments. The first argument is the starting row number, the second argument is the number of rows to delete. In the following example, rows 6 through 8 in `panel_list_item` are removed. Row 0 is the first row.

```
xv_set(panel_list_item,
    PANEL_LIST_DELETE_ROWS, 6, 2,
    NULL);
```

If you use `PANEL_LIST_DELETE` to delete multiple list items, you need to delete list items in descending order. When a row is deleted, the row numbers are adjusted to a sequential order. For example, to delete rows `1` through `5`:

```
int row;
for (row = 5; row >=1; row--)
    xv_set(panel_list_item, PANEL_LIST_DELETE, row, NULL);
```

To add to a scrolling list, starting at a particular row, you can use `PANEL_LIST_INSERT_STRINGS` or `PANEL_LIST_INSERT_GLYPHS`. `PANEL_LIST_INSERT_STRINGS` inserts strings into a specified scrolling list before a specified row. `PANEL_LIST_INSERT_GLYPHS` inserts glyphs into a specified scrolling list before a specified row.

*See Appendix D for a description of an improved list insertion method that is available in XView Version 3.2. and newer releases.

To add a new row, PANEL_LIST_INSERT is used in the same way as PANEL_LIST_DELETE. When adding a new row in this manner, all the succeeding row numbers are incremented and the list size grows by one. Space for a new item is created, and a new string or glyph may be added. These are all done at the time the attribute is evaluated, so they may be combined into one xv_set() call. You can move a row by deleting it from its old location and reassigning it to a new location. Look at the following code:

```
char *buf[128];

strcpy(buf, xv_get(list_item, PANEL_LIST_STRING, 4));

xv_set(list_item,
    PANEL_LIST_DELETE,    4,
    PANEL_LIST_INSERT,    8,
    PANEL_LIST_STRING,    8, buf,
    NULL);
```

The value for the string *must* be copied because as soon as the list item is deleted, the data is freed.

If a panel list may *not* contain duplicate entires, PANEL_LIST_INSERT_DUPLICATE needs to be set to FALSE. The default value for this attribute is TRUE, which allows duplicate strings to be inserted.

7.11.3 List Selection

Items in the list are selected by using the SELECT mouse button while pointing at an item or by dragging the pointer over the list items, or with the attribute PANEL_LIST_SELECT. When PANEL_LIST_SELECT is used to select an item that is currently visible, then the list may be scrolled when PANEL_LIST_SELECT is set. To disable the scrolling, set XV_SHOW to FALSE for the scrolling list before the specified row is selected with PANEL_LIST_SELECT.

Selected choice(s) can be set at list creation or later by using PANEL_LIST_SELECT. For example:

```
PANEL_LIST_SELECT, 3, TRUE,
```

will select row three. If the list item is nonexclusive, you can set more than one choice at one time. For example:

```
PANEL_LIST_SELECT, 3,  TRUE,
PANEL_LIST_SELECT, 13, TRUE,
PANEL_LIST_SELECT, 14, TRUE,
```

will select rows 3, 13, and 14.

To determine if a row is selected, use:

```
xv_get(list_item, PANEL_LIST_SELECTED, i);
```

This call to `xv_get()` returns `TRUE` if row *i* is selected. To return the first selected row, use `PANEL_LIST_FIRST_SELECTED` as follows:

```
int first_selected;
first_selected = (int)xv_get(list_item, PANEL_LIST_FIRST_SELECTED);
```

`PANEL_LIST_NEXT_SELECTED` returns the next selected row after a specified row. This attribute takes a single integer argument representing the row to start from. If no row is selected following the specified row, `PANEL_LIST_NEXT_SELECTED` returns -1.

7.11.4 List Notification

The procedure specified via the attribute `PANEL_NOTIFY_PROC` is called when a row is selected, de-selected, added, or deleted. List notify procedures are passed the following: the list item, the string indicating the label of the item being acted upon, any client data associated with the list entry, a parameter indicating the action being taken, the event which caused notification, and the row number of the row being operated on. The form of the procedure is:

```
int
list_notify_proc(item, string, client_data, op, event, row)
    Panel_item          item; /* panel list item */
    char                *string;
    Xv_opaque           client_data;
    Panel_list_op       op;
    Event               *event;
    int                 row; /* row number */
```

`item` is the panel list item that contains the row that was acted upon. The `string` parameter is the label of the row. If there is no string associated with the row, the parameter is `NULL`. If the row contains both a string and an image, then the string is passed as the parameter.

`op` is one of the following:

```
PANEL_LIST_OP_SELECT
PANEL_LIST_OP_DESELECT
PANEL_LIST_OP_VALIDATE
PANEL_LIST_OP_DELETE
```

If the user selects a row that is not currently selected, the notify procedure is called twice. The first time it is called with the previously selected row, the `op` is `PANEL_LIST_OP_DESELECT`. The next time the function is called, the `op` is `PANEL_LIST_OP_SELECT`. If the user selects an item that is already selected, the function is called once, passing the row selected (`op` is `PANEL_LIST_OP_SELECT`). Validate is called when the user inserts a new row (*op* is set to `PANEL_LIST_OP_VALIDATE`). The notify procedure should return `XV_OK` to accept the row, or `XV_ERROR` to reject the row. Delete is called when the user deletes a row (*op* is set to `PANEL_LIST_OP_DELETE`).

7.11.4.1 List item client data

The `client_data` parameter is set to whatever client data is associated with the row. *list_glyphs.c* uses the attribute `PANEL_LIST_CLIENT_DATAS` to assign a set of values to the list items. Each value could have been assigned to the rows one by one using the attribute `PANEL_LIST_CLIENT_DATA`. This attribute takes two values: the first is the number of the row to assign the data to, and the second is the data itself.

```
xv_create(panel, PANEL_LIST,
    ...
    PANEL_LIST_CLIENT_DATA, 0, "one",
    PANEL_LIST_CLIENT_DATA, 1, "two",
    PANEL_LIST_CLIENT_DATA, 2, "three",
    ...
    NULL);
```

You can still assign client data to the panel list item itself using `XV_KEY_DATA` as with any other XView object. However, this data can only be retrieved using `xv_get()` from the panel list item itself, the first parameter in the callback function.

7.11.5 The Scrolling List Menu

`PANEL_ITEM_MENU` sets or gets the scrolling list's read menu if the Scrolling List is in read mode or the edit menu if the Scrolling List is in edit mode. The mode of the scrolling list is set with the attribute `PANEL_LIST_MODE`. `PANEL_LIST_MODE` takes one of two values: `PANEL_LIST_READ` and `PANEL_LIST_EDIT`. The attribute `PANEL_VALUE_STORED_LENGTH` controls the amount of text that may be edited when in edit mode.

7.12 Message Items

Message items display a text or image message within a panel. The only visible component of a message item is the label itself. Message items are useful for annotations of all kinds, including titles, comments, descriptions, pictures and dynamic status messages. The message is often used to identify elements on the panel. A message has no value.

You may set or change the label for a message item via `PANEL_LABEL_STRING` or `PANEL_LABEL_IMAGE`. Message items can have notify procedures which are called when `SELECT_UP` occurs over the message item. Panel message items are the only panel items whose font can be set to boldface. The boldness of message items is controlled using the attribute `PANEL_LABEL_BOLD`.

Since messages cannot be selected, their primary use is for display purposes only. In Example 7-4, two message items display normal pipeline pressure, and a warning for high pipeline pressure.

Example 7-4. The message_item.c program

```
#include <xview/xview.h>
#include <xview/panel.h>

Frame         frame;
Panel_item    message;
Panel         panel;
Panel_item    slider;


void
slider_notify_proc(item, value, event)
    Panel_item      item;
    int             value;
    Event    *event;
{
    xv_set(message,
     PANEL_LABEL_STRING,
         value > 500 ? "** HIGH PIPELINE PRESSURE **" : "Okay",
     0);
}


main(argc, argv)
    int        argc;
    char  **argv;
{
    xv_init(XV_INIT_ARGS, argc, argv, 0);

    frame = xv_create(NULL, FRAME,
                FRAME_LABEL,    "Message Item",
                0);

    panel = xv_create(frame, PANEL,
     PANEL_LAYOUT, PANEL_VERTICAL,
     0);

    slider = xv_create(panel, PANEL_SLIDER,
     PANEL_LABEL_STRING, "Pipeline pressure (psi):",
     PANEL_MIN_VALUE, 0,
     PANEL_MAX_VALUE, 1000,
     PANEL_NOTIFY_PROC, slider_notify_proc,
     PANEL_SHOW_RANGE, TRUE,
     PANEL_VALUE, 100,
     0);
    message = xv_create(panel, PANEL_MESSAGE,
     PANEL_LABEL_STRING, "Okay",
     0);

    window_fit(panel);
    window_fit(frame);

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

Messages produced by this program are shown in Figure 7-15 and Figure 7-16.

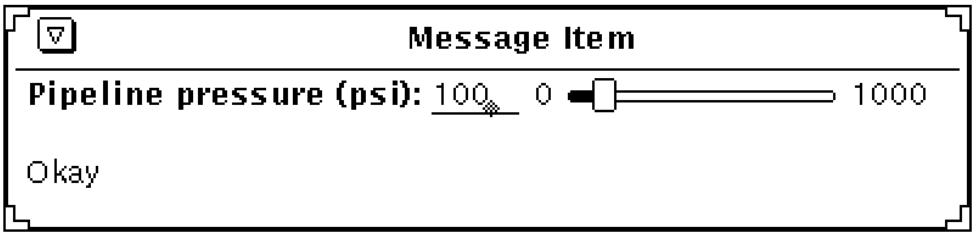

Figure 7-15. Sample panel with message item

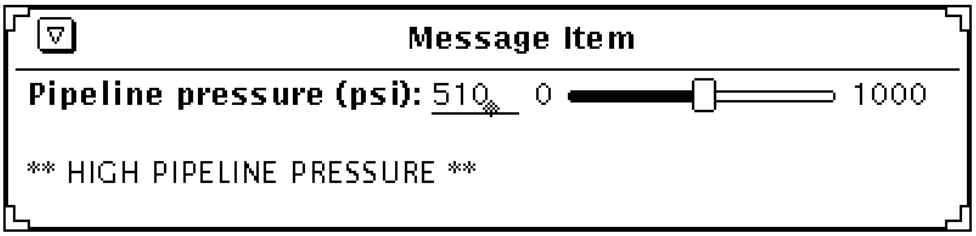

Figure 7-16. Sample panel with message item–High Pressure

7.13 Slider Items

Slider items allow the graphical representation and selection of a value within a range. Sliders are appropriate for situations where it is desired to make fine adjustments over a continuous range of values. The user selects the slider bar and drags it to the value that he wishes. A slider has the following displayable components: the label, the current value, the slider bar and the minimum and maximum allowable integral values (the range), end boxes, tick marks, tick mark minimum and maximum tick strings, as well as minimum and maximum value text strings.

Sliders may be horizontal or vertical depending on the value of `PANEL_DIRECTION`. This attribute defaults to `PANEL_HORIZONTAL`, but may be set to a vertical orientation by using the value `PANEL_VERTICAL`. The attribute `PANEL_SLIDER_END_BOXES` sets whether the boxes at the endpoints of the slider are visible. This attribute defaults to `FALSE`. The `PANEL_TICKS` attribute takes a numeric value that indicates how many evenly spaced “tick-marks” are drawn next to the item. When `PANEL_SHOW_VALUE` is `TRUE`, the current value is shown after the label in an editable text field. The minimum and maximum allowable values are set with `PANEL_MIN_VALUE` and `PANEL_MAX_VALUE`. The width of the slider bar can be adjusted using the `PANEL_SLIDER_WIDTH` attribute. When `PANEL_SHOW_RANGE` is `TRUE`, the minimum value of the slider `PANEL_MIN_VALUE` is shown to the left of the slider bar and the maximum value `PANEL_MAX_VALUE` is shown to the right of the slider bar.* The attributes `PANEL_MIN_TICK_STRING` and `PANEL_MAX_TICK_STRING` specify text labels for the minimum and maximum tick values. On horizontal sliders, these strings appear underneath the maximum and minimum tick marks. If the attribute `PANEL_SHOW_RANGE` does not adequately describe the slider values, `PANEL_MIN_VALUE_STRING` and

*The top and bottom of the slider is used when the orientation is vertical.

PANEL_MAX_VALUE_STRING can specify string value labels for the minimum and maximum slider values. On horizontal sliders, these strings appear to the left/right of the minimum/maximum end boxes.

7.13.1 Slider Selection

Only the slider bar of a slider may be selected. When the SELECT button is pressed within the slider drag box, the black area of the bar will advance or retreat with the position of the cursor. The slider value can also be changed via the numeric text field by clicking in the slider bar to the left (horizontal sliders) or below (vertical sliders) the drag box to decrement the value, or by clicking in the slider bar to the right (horizontal sliders) or above (vertical sliders) to increment the value.

7.13.2 Slider Notification

Slider notify procedures are passed the item, the item's value at time of notification, and the event which caused notification:

```
void
slider_notify_proc(item, value, event)
    Panel_item  item;
    int         value;
    Event      *event;
```

The notification behavior of a slider is controlled by the value of PANEL_NOTIFY_LEVEL. It can be set to one of two values:

PANEL_DONE — The default. The notify procedure is called only when the SELECT button is released within the panel or when the user types in a new value for the slider's numeric text field.

PANEL_ALL — The notify procedure is called whenever the value of the slider is changed; this includes when the user selects, drags or releases the SELECT button in a slider. For each movement of the mouse while dragging the slider drag box, the slider's notify procedure is called.

7.13.3 Slider Value

The value of a slider is an integer in the range PANEL_MIN_VALUE to PANEL_MAX_VALUE. You can retrieve or set a slider's value with the attribute PANEL_VALUE and the functions xv_set() or xv_get().

The following code fragment produces a slider with a label:

```
xv_create(panel, PANEL_SLIDER,
    PANEL_LABEL_STRING,    "Brightness: ",
    PANEL_VALUE,           75,
    PANEL_MIN_VALUE,       0,
    PANEL_MAX_VALUE,       100,
    PANEL_SLIDER_WIDTH,    300,
    PANEL_TICKS,           5,
    PANEL_NOTIFY_PROC,     brightness_proc,
    NULL);
```

The output is shown in Figure 7-17.

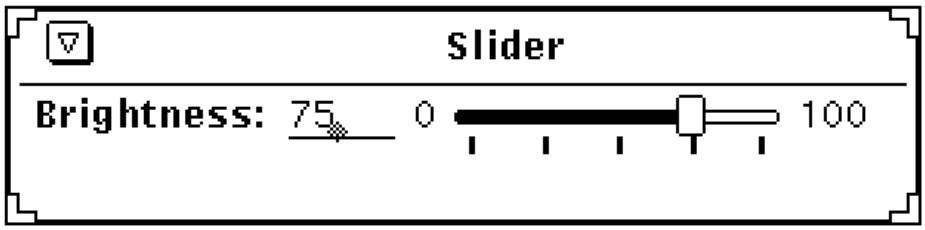

Figure 7-17. Sample panel with slider item

7.14 Gauges

Gauges are just like sliders, but they are “output only” items. That is, you set the value of the item and the display of the gauge changes just as it would for sliders. Also, there is no optional type-in field and there is no drag box for the user to interactively change the value of the gauge. The gauge is intended to be used only as a feedback item.

To create a gauge, use the `PANEL_GAUGE` package. To set a gauge’s width or height, use `PANEL_GAUGE_WIDTH`. This attribute sets the length of the object, whether it is vertically or horizontally oriented. As with the slider, the orientation is set by the attribute `PANEL_DIRECTION`.

7.15 Text Items

A panel text item contains as its value a `NULL`-terminated string. It contains only printable characters with no newlines. When a panel receives keyboard input (regardless of where the pointer is as long as it is within the boundaries of the panel), the keyboard event is passed to the item with the keyboard focus. A caret is used to indicate the insertion point where new text is added. You can type in more text than fits on the text field. If this happens, a right arrow pointing to the left will appear on the left on the field, indicating that some text to the left of the displayed text is no longer visible. Similarly, if text is inserted causing text on the

right to move out of the visible portion of the text item, then an arrow pointing to the right will appear to the right of the text.

Text items use the attribute PANEL_LABEL_STRING as do most other panel items, to label the text item. The value of a text item is also a string, an is set by the attribute PANEL_VALUE, as shown by the following code:

```
xv_create(panel, PANEL_TEXT,
    PANEL_LABEL_STRING,  "Name:",
    PANEL_VALUE,         "Edward G. Robinson",
    NULL);
```

The output from this code fragment is shown in Figure 7-18.

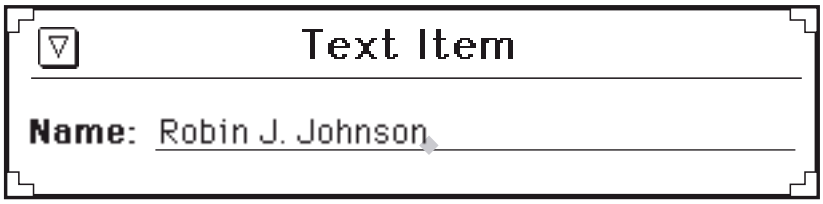

Figure 7-18. Sample panel with text item

If the item's PANEL_LAYOUT is set to PANEL_VERTICAL, the value is placed below the label. The default is PANEL_HORIZONTAL.

The number of characters of the text item's value that are displayed is set via PANEL_VALUE_DISPLAY_LENGTH. Note that the length of the value display specified by PANEL_VALUE_DISPLAY_LENGTH may not be less than the combined width of the left and right "more text" buttons. In 12-point font, this is four characters.

PANEL_VALUE_DISPLAY_LENGTH is useful for fixed width characters. PANEL_VALUE_DISPLAY_LENGTH converts characters to pixels by multiplying the number of characters by the default character width for the font being used. To set the number of characters in the text item's value that are displayed in a variable width font, use PANEL_VALUE_DISPLAY_WIDTH. The argument for PANEL_VALUE_DISPLAY_WIDTH is expressed in pixels instead of characters. The maximum number of characters that can be typed into a text item (independently of how many are displayable) is set via the attribute PANEL_VALUE_STORED_LENGTH. When characters are entered beyond the display length, and when PANEL_VALUE_STORED_LENGTH is greater than the display length, the value string is scrolled one character to the left so that the most recently entered character is always visible. As the string scrolls to the left, the leftmost characters move out of the visible display area. The presence of these temporarily hidden characters is indicated by a small left-pointing triangle.

It is sometimes desirable to have a protected field where the user can enter confidential information. The attribute PANEL_MASK_CHAR is provided for this purpose. When the user enters a character, the character specified as the value of PANEL_MASK_CHAR will be displayed in place of the character the user has typed. Setting PANEL_MASK_CHAR to an asterisk (*) would

produce a string of asterisks instead of the characters typed. The value of the text is still the string the user types.

If you want to disable character echo entirely so that the caret does not advance and it is impossible to tell how many characters have been entered, use the space character as the mask. You can remove the mask and display the actual value string at any time by setting the mask to `NULL`.

7.15.1 The Current Keyboard Focus

A panel may have several keyboard focus items that can accept keyboard input. Only one of these items may be *current* at any given time. The current keyboard focus item is the one to which keyboard input is directed and is indicated by a caret at the item's value. Selection of a keyboard focus item (i.e., pressing and releasing the SELECT mouse button anywhere within the item's bounding box) causes that item to become the current keyboard focus item.

NOTE

When the resource `OpenWindows.KeyboardCommands` is set to SunView1 or Basic, only text items will receive the keyboard focus within a panel.

You can find out which keyboard focus item has the caret or give the caret to a specified keyboard focus item by means of the panel attribute `PANEL_CARET_ITEM`:

```
current_item = (Panel_item)xv_get(panel, PANEL_CARET_ITEM);
xv_set(panel, PANEL_CARET_ITEM, another_item, NULL);
```

You can set the current item to the next or previous keyboard focus item in the panel by using the following two routines:

```
Panel_item
panel_advance_caret(panel)
    Panel panel;

Panel_item
panel_backup_caret(panel)
    Panel panel;
```

They return the new panel item that has received the keyboard focus. Advancing past the last keyboard focus item places the keyboard focus at the first keyboard focus item, while backing up past the first keyboard focus item places the keyboard focus at the last keyboard focus item.

7.15.2 Text Selection

You can use the attribute PANEL_TEXT_SELECT_LINE to select and highlight the entire contents of the text field.

7.15.3 Text Notification

The notification behavior of text items is more complex than that of other item types. You can control whether your notify procedure is called on each input character or only on selected characters.

When your notify procedure will be called is determined by the value of PANEL_NOTIFY_LEVEL. Possible values are given in Table 7-1.

Table 7-1. Text Item Notification Level

Notification Level	Level Causes Notify Procedure to be Called . . .
PANEL_NONE	Never
PANEL_NON_PRINTABLE	On each non-printable input character.
PANEL_SPECIFIED	If the input char is found in the string given for the attribute PANEL_NOTIFY_STRING.
PANEL_ALL	On each input character.

The default for PANEL_NOTIFY_LEVEL is PANEL_SPECIFIED, and the default for PANEL_NOTIFY_STRING is \n\r\t (i.e., notification on line-feed, carriage-return and tab). The value PANEL_SPECIFIED only works for characters that do *not* map to semantic actions. To provide notification for characters that map to semantic actions, such as 177 (delete normally maps to ACTION_ERASE_CHAR_BACKWARD), use either PANEL_EVENT_PROC, notify_interpose_event_func(), or set PANEL_NOTIFY_LEVEL to PANEL_ALL.

The user's editing characters are treated specially (for example the backspace character). They are *not* appended to the value string. If you have asked for a character by including it in PANEL_NOTIFY_STRING, the PANEL package calls your notify procedure. After the notify procedure returns, the appropriate editing operation will be applied to the value string. (Note that the editing characters are never appended to the value string, regardless of the return value of the notify procedure.)

Characters other than the special characters described above are treated as follows. If the character typed by the user does *not* result in your notify procedure getting called, then the character, if printable, is appended to the value string. If it is not printable, it is ignored. If your notify procedure *is* called, what happens to the value string and whether the keyboard focus moves to another item is determined by the notify procedure's return value. Table 7-2 shows the possible return values.

Table 7-2. Return Values for Text Item Notify Procedures

Value Returned	Action Caused
PANEL_INSERT	Character is appended to item's value.
PANEL_NEXT	Keyboard focus moves to next text item.
PANEL_PREVIOUS	Keyboard focus moves to previous text item.
PANEL_NONE	Ignore the input character.

If you do not specify your own notify procedure, the default procedure, `panel_text_notify()`, is called at the appropriate time as determined by the setting of PANEL_NOTIFY_LEVEL.

7.15.4 Writing Your Own Text Notify Procedure

By writing your own notify procedure, you can tailor the notification behavior of a given keyboard focus item to support the needs of any application. At one extreme, you may want to process each character as the user types it in. For a different application, you may not care about the characters as they are typed in and may only want to look at the value of the string in response to some other button. A typical example is getting the value of a filename field when the user presses a **Load File** panel button.

The form of the text notification procedure is:

```
Panel_setting
panel_text_notify(item, event)
    Panel_item  item;
    Event      *event;
```

This procedure returns a panel setting enumeration that has the following effects:

PANEL_NONE — Do not advance the keyboard focus to the next keyboard focus item. The current keyboard focus item and insertion point remain unchanged.

PANEL_NEXT — The keyboard focus moves to the *next* keyboard focus item (defined by the keyboard focus item that was created *after* the current keyboard focus item). If there is no next text item, the first keyboard focus item in the panel is used.

PANEL_PREVIOUS — The keyboard focus moves to the *previous* keyboard focus item in the panel (defined by the keyboard focus item that was created *before* the current keyboard focus item). If there is no previous keyboard focus item, the last keyboard focus item in the panel is used.

PANEL_INSERT — The character which caused the notification procedure to be called is inserted into the text item's value at the location of the caret (insertion point).

7.15.5 Text Value

You can set or get the value of a keyboard focus item at any time via PANEL_VALUE. The following call retrieves the value of name_item into name:

```
Panel_item name_item;
char name[NAME_ITEM_MAX_LENGTH];
        ...
strcpy(name, (char *)xv_get(name_item, PANEL_VALUE));
```

Note that name_item should have been created with a PANEL_VALUE_STORED_LENGTH not greater than NAME_ITEM_MAX_LENGTH so the buffer name will not overflow.

7.16 Numeric Text Items

Panel numeric text items are virtually the same as panel text items except that the value displayed is of type int. Also, convenience features (such as increment and decrement buttons) ease the manipulation of the text string's numeric value. There is little programmatic difference between the text item and the numeric text item. You create a numeric text item using the PANEL_NUMERIC_TEXT package. You can also set the minimum and maximum range for the numeric text field by using PANEL_MIN_VALUE and PANEL_MAX_VALUE, respectively.

7.17 Multiline Text Items

Multiline text items are a special type of panel text item that allow a text field to display multiple lines. You create a multiline text item using the PANEL_MULTILINE_TEXT package. Multiline text items use the attribute PANEL_DISPLAY_ROWS to specify the number of rows of text to display. PANEL_VALUE_DISPLAY_LENGTH specifies the length in characters of a row in a multiline text field. PANEL_VALUE_DISPLAY_WIDTH specifies the length in pixels of each row. A multiline text item will have scrollbars if the stored length is greater than the displayed length (rows × columns). The maximum stored length for a multiline text item is specified using PANEL_VALUE_STORED_LENGTH.* If PANEL_LINE_BREAK_ACTION is PANEL_WRAP_AT_CHAR, the lines wrap as soon as the number of characters on a line exceeds the length of the line. If this attribute is PANEL_WRAP_AT_WORD, the lines in the multiline text item wrap only at word breaks.

*XView Version 3 *only* supports multiline text items with scrollbars.

NOTE

Multiline text items display multiple rows, or “lines,” but do not contain embedded returns or line feeds.

Example 7-5 shows how to create a multiline text item with scrollbars.

Example 7-5. The multiline.c program

```
/*
 * multiline.c -- simple panel multiline text item example.
 */
#include <xview/xview.h>
#include <xview/panel.h>

Frame frame;
Panel panel;

main(argc, argv)
    int argc;
    char **argv;
{

    xv_init(XV_INIT_ARGS, argc, argv, NULL);

    frame = xv_create(NULL, FRAME, NULL);
    panel = xv_create(frame, PANEL, NULL);
    xv_create(panel, PANEL_MULTILINE_TEXT,
        PANEL_LABEL_STRING, "Product Description:",
        PANEL_DISPLAY_ROWS, 6,
        PANEL_VALUE_DISPLAY_LENGTH, 32,
        PANEL_VALUE, "This wonderful product is \
designed to allow the user to combine and manipulate both \
text and graphic objects easily. The goal of the design team \
is to provide an intuitive, logical interface.",
        NULL);

    window_fit(panel);
    window_fit(frame);
    xv_main_loop(frame);
}
```

The output of this code fragment is shown in Figure 7-19.

7.18 Drop Target Items

A panel *drop target item* is a bordered image in a panel area that is used to transfer data between applications. Before you use a panel drop target item you need to be familiar with the `SELECTION` and `DRAGDROP` packages; these are described in Chapter, 18, *Selections*, and Chapter 19, *Drag and Drop*.

A panel drop target item is an object in the class `Panel_drop_target_item` which is equivalent to a `Panel_item`. A drop target item’s owner is a `Panel`. Examples of several drop target items are shown in Figure 7-20.

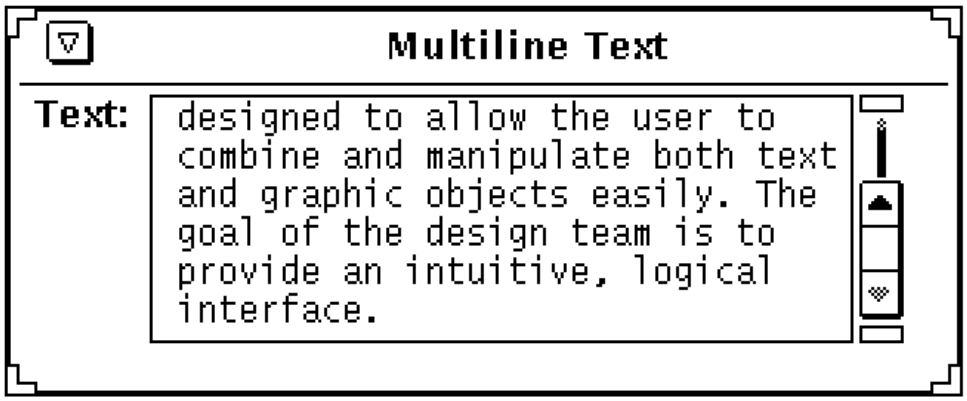

Figure 7-19. Panel multiline text item

To use the PANEL_DROP_TARGET package in an application, you need to include both the <*xview/dragdrop.h*> and <*xview/svrimage.h*> header files. The program *panel_dnd.c*, showing a panel drag and drop example, is presented in Appendix F, *Example Programs.*

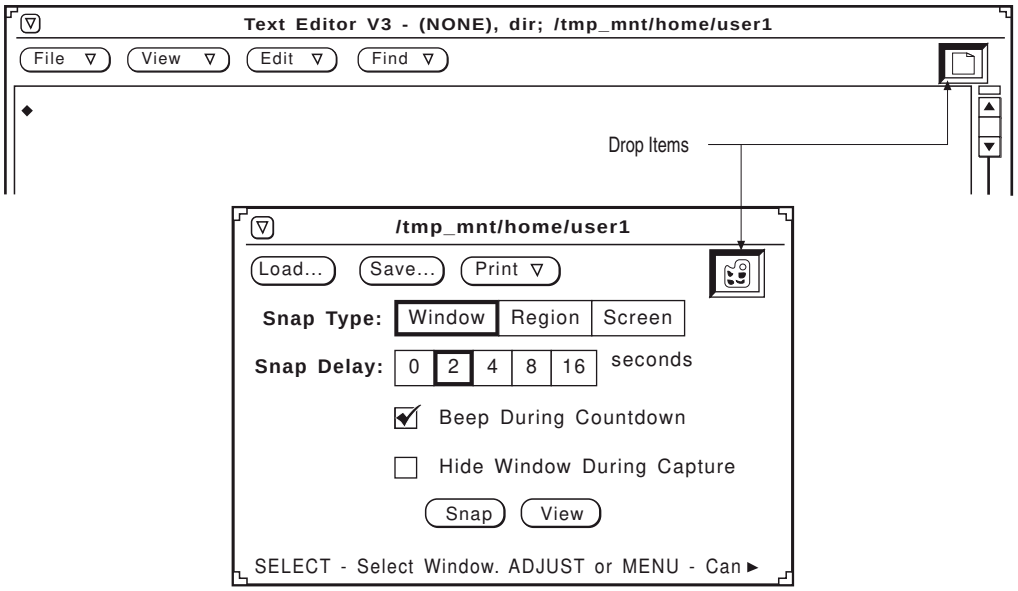

Figure 7-20. Sample panel with drop target items

7.18.1 Programming a Panel Drop Target Item

To use a panel drop target item follow these programming steps:

- Create the drop target item.
- Specify the glyphs.
- Create drag and drop object.
- Define the drop target item's requestor.
- Control the glyphs.
- Drop on the target item.
- Drag from the drop target item.

7.18.1.1 Create the drop target item

You create a panel drop target using `xv_create()` with the `PANEL_DROP_TARGET` package. The following code fragment shows a how to create a drop target item called `drop_target`:

```
Panel_drop_target_item       drop_target;
    drop_target = xv_create(panel, PANEL_DROP_TARGET, NULL);
```

7.18.1.2 Specify the glyphs

The attribute `PANEL_DROP_GLYPH` specifies the glyph for a "normal" drop target. The normal glyph is shown when the drop target item is inactive (no data transfer is occurring). `PANEL_DROP_BUSY_GLYPH` specifies the glyph for the "busy" drop target. The busy drop target glyph is displayed when the drop target item is receiving a drop, or when data is being sent to another application. You can define the bits for the glyphs as follows (assuming the files *normal.icon* and *busy.icon* contain appropriate data):

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

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

Once the bits are defined, you create server images to represent them as follows:

```
normal_glyph = xv_create(NULL, SERVER_IMAGE,
    XV_HEIGHT, 64,
    XV_WIDTH, 64,
    SERVER_IMAGE_DEPTH, 1,
    SERVER_IMAGE_BITS, normal_bitmap,
    NULL);
```

```
busy_glyph = xv_create(NULL, SERVER_IMAGE,
    XV_HEIGHT, 64,
    XV_WIDTH, 64,
    SERVER_IMAGE_DEPTH, 1,
    SERVER_IMAGE_BITS, busy_bitmap,
    NULL),
```

Set the panel drop target attributes to use the server images defined above:

```
xv_set(drop_target,
    PANEL_DROP_GLYPH, normal_glyph,
    PANEL_DROP_BUSY_GLYPH, busy_glyph,
    NULL);
```

7.18.1.3 Create the drag and drop object

If the drop target item will support drags, create a `Drag_drop` object and set the attribute `PANEL_DROP_DND`. This attribute is the `DRAGDROP` object associated with the panel drop target item. The `Drag_drop` object is used to initiate a drag and drop operation. If `PANEL_DROP_DND` does not exist, then the panel drop target item will not support drags and is called an *empty* drop target. In this case, `PANEL_DROP_FULL` will be `FALSE` (the default).

```
Drag_drop dnd;
    dnd = xv_create(panel, DRAGDROP, NULL);
```

If the drop target item will support drags, then set the appropriate attributes on the `Dnd` object (rank, cursor, etc.).

Create a selection item associated with the `Dnd` object. This defines the data and the conversion(s) supported for the source of the drag. For example:

```
xv_create(dnd, SELECTION_ITEM,
        SEL_DATA, "dnd selection data",
        NULL);
```

After data is defined, you need to set `PANEL_DROP_FULL` to `TRUE`. When set to `TRUE`, `PANEL_DROP_FULL` indicates that valid, “draggable” data is set on the `PANEL_DROP_DND` object’s selection items. For example:

```
xv_set(drop_target,
    PANEL_DROP_DND,  dnd,
    PANEL_DROP_FULL, TRUE,
    NULL);
```

7.18.1.4 Define the drop target item’s requestor

The panel package creates a selection requestor, from the `SELECTION_REQUESTOR` package that is associated with each drop target item. This selection requestor’s attributes need to be set. `PANEL_DROP_SEL_REQ` returns the `SELECTION_REQUESTOR` associated with the panel drop target item. The following code fragment gets the selection requestor associated with an item:

```
Selection_requestor sel_req;
```

```
sel_req = xv_get(item, PANEL_DROP_SEL_REQ);
```

7.18.1.5 Controlling the glyphs

The drop target item package handles this step. When something is dragged into the drop target box, the busy glyph is displayed. This action is initiated when an ACTION_DRAG_PREVIEW semantic action and LOC_WINENTER event id combination is received on the drop target box. When the cursor is dragged out of the drop target box, the glyph changes back to the normal state. An ACTION_DRAG_PREVIEW semantic action and LOC_WINEXIT event id combination on the drop target box initiates this.

7.18.1.6 Dropping on the drop target

When something is dropped on the drop target item, the panel package calls dnd_decode_drop(). This action is initiated by an ACTION_DRAG_COPY or ACTION_DRAG_MOVE. When dnd_decode_drop() returns, the panel drop target item's notify procedure is called. For details, see Section 7.18.2, "Drop Target Notification."

7.18.1.7 Dragging from the drop target item

When SELECT is pressed while over the drop target item, if PANEL_DROP_FULL is TRUE, dnd_send_drop() is called. At this time the glyph is changed to its busy state. When dnd_send_drop() returns, the panel drop target item's notify procedure is called.

7.18.2 Drop Target Notification

When the user drops an item on the panel drop target item, the item and the event that initiated the drop are passed to the notify procedure. The form of a panel-drop-target notify procedure is:

```
int
drop_target_notify_proc(item, value, event)
   Panel_drop_target_item  item;
   int                     value
   Event                  *event;
```

The *item* is the panel drop target that was dropped on. The argument *value* contains the return value from dnd_decode_drop(). The *event* is the event which initiates the drop (i.e., ACTION_DRAG_COPY or ACTION_DRAG_MOVE).

When the notify procedure returns, the glyph is returned to its normal state. The notify procedure returns one of the following: XV_OK or XV_ERROR. When the notify procedure returns, one of the following actions occurs:

1. If the return value is XV_OK, then the function dnd_done() is called.
2. If the return value is XV_ERROR, then the function dnd_done() is *not* called.

7.19 Advanced Panel Usage

The following sections address some advanced topics dealing with panels. They cover attaching data to panel items, repainting panels and panel items, and handling events in panels and in panel items. Handling panel repainting and panel events are features which are available but are not generally used by most applications. Attaching data to panel items should be a practice closely followed for more efficient programs.

7.19.1 Attaching Data to Panel Items

Callback routines are called separately and independently from the application's main routine. If the callback routine needs data, there are two ways to make it available. One way is to store the data in global variables or data structures so the callback routine can reference the data. Another way is to attach the data directly to panel items (whose handle has already been retrieved by the notification procedure).

In the spirit of good programming practice, it is wise to avoid creating global variables. The preferred method for making data available to callback routines is to *attach* the data to the panel items. A handle to the panel item is already made available to the callback function as the first parameter to the function. Two attributes can be used to attach data to panel items: `XV_KEY_DATA` or `PANEL_CLIENT_DATA`.*

Using `XV_KEY_DATA` requires a *key*, which must be some arbitrary, but unique, integer. An example of usage follows:

```
static int My_item_key = 0;
...
if (!My_item_key) My_item_key =  xv_unique_key();
xv_create(panel, PANEL_BUTTON,
    PANEL_BUTTON_LABEL,    "Push Me",
    XV_KEY_DATA,           My_item_key, "text",
    PANEL_NOTIFY_PROC,     my_notify_proc,
    ...
    NULL);
```

If this method is used, `my_notify_proc()` retrieves the data using `xv_get()`:

```
char *data = (char *)xv_get(item, XV_KEY_DATA, My_item_key);
```

The common element in this case is the use of `My_item_key`. A `static int` need not be used, but it is better than creating a global `int` variable to store the key. If that were the case, you might as well make the *data* portion that you wanted to attach to the panel item a global variable.

One advantage to using `XV_KEY_DATA` is that you can specify any number of keys and attach as many pieces of data to objects as you like.

*`XV_KEY_DATA` can be used with any XView object. Client data can as well, but the name to use varies with the object package, e.g., `WIN_CLIENT_DATA` is for windows, `MENU_CLIENT_DATA` is for menus, etc.

An alternate method for attaching data to panel items is to use PANEL_CLIENT_DATA. This attribute is similar to XV_KEY_DATA in that the data is attached to the item, but in this case, you can only attach one piece of data to the panel item. You can, however, use *both* XV_KEY_DATA and PANEL_CLIENT_DATA on the same panel item. The advantage to using PANEL_CLIENT_DATA is that you do not have to keep track of a key. Since you can only attach one piece of data to a panel item, `xv_get()` returns only that data.

For a real example of this situation, let's modify the *quit.c* program (from the beginning of this chapter), which displays a frame, panel and a panel item labeled Quit. The modified program, *client_data.c*, is shown in Example 7-6. Selecting the panel button exits the program gracefully. The call to `xv_destroy_safe(frame)` causes the frame to be destroyed and thus, `xv_main_loop()` returns and the program exits.

Example 7-6. The client_data.c program

```
/*
 * client_data.c -- demonstrate the use of PANEL_CLIENT_DATA attached
 * to panel items.  Attach the base frame to the "Quit" panel item so
 * that the notify procedure can call xv_destroy_safe() on the frame.
 */
#include <xview/xview.h>
#include <xview/panel.h>

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

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

    frame = (Frame)xv_create(XV_NULL, FRAME, NULL);
    panel = (Panel)xv_create(frame, PANEL, NULL);
    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Quit",
        PANEL_NOTIFY_PROC,      quit,
        PANEL_CLIENT_DATA,      frame,
        NULL);

    xv_main_loop(frame);
    puts("The program is now done.");
    exit(0);
}

quit(item)
Panel_item item;
{
    Frame frame = (Frame)xv_get(item, PANEL_CLIENT_DATA);
    xv_destroy_safe(frame);
}
```

In this program, the `frame` object is not a global variable; it is a local, or *automatic*, variable. Since there is a close association between the panel button and the frame (meaning that the panel button is going to access the handle to the frame in the callback routine), we attach

the frame to the panel button as *client data*. PANEL_CLIENT_DATA takes a generic address of type caddr_t. In the callback routine for the panel button, the first parameter to the callback function is the panel item that called the notification. From that handle, the frame is retrieved via xv_get().

This worked because the frame was created via xv_create(). That is, the object was *allocated*. You cannot attach data that has not been allocated. Thus, the following code segment is not advised:

```
dummy_function()       /* Wrong way */
{
    char *home = (char *)getenv("HOME");
    xv_set(panel_item, XV_KEY_DATA, HOME_KEY, home, NULL);
}
```

The reason for this is that getenv() returns a pointer to static data that is *overwritten on each call*. The next call that the application makes to getenv() will change the value for the panel item XV_KEY_DATA. Likewise, the following should not be used:

```
dummy_function()     /* Also wrong */
{
    char home[MAXPATHLEN], *ptr;
    if ((ptr = (char *)getenv("HOME")) != NULL) {
        (void) strcpy(home, ptr);
        xv_set(panel_item, XV_KEY_DATA, HOME_KEY, home, NULL);
    }
}
```

This does not work because as soon as dummy_function() returns, the value of home is lost because it is an automatic variable. The correct way to handle this is to make home either a static variable or a pointer whose storage is allocated via malloc().

The problem with home being static is that if dummy_function() is called more than once, the value of home will be overwritten on each call. So, the *best* way to handle this case is to allocate the data:

```
dummy_function()     /* Best way */
{
    extern char *malloc(), *getenv();
    char *home, *ptr;
    if ((ptr = getenv("HOME")) != NULL &&
        (home = malloc(strlen(ptr)+1))) {
        (void) strcpy(home, ptr);
        xv_set(panel_item, XV_KEY_DATA, HOME_KEY, home, NULL);
    }
}
```

Having allocated data for XV_KEY_DATA, we now assume the responsibility of freeing that data when the object is destroyed. Otherwise, the data is left free with no references to it. This is also known as creating a *memory leak*. Because you don't always know when an object is being destroyed (destroying a panel may or may not cause a panel item to be destroyed), you can specify a function that explicitly frees the data pointed to by XV_KEY_DATA. To do this, use the attribute XV_KEY_DATA_REMOVE_PROC:

```
dummy_function()
{
    extern void free_data();
```

```
        extern char *malloc(), *getenv();
        char *home, *ptr;
        if ((ptr = getenv("HOME")) != NULL &&
            (home = malloc(strlen(ptr)+1))) {
            (void) strcpy(home, ptr);
            xv_set(panel_item,
                XV_KEY_DATA,             HOME_KEY, home,
                XV_KEY_DATA_REMOVE_PROC, HOME_KEY, free_data,
                NULL);
        }
    }

    void
    free_data(object, key, data)
    Xv_object object;
    int       key;
    caddr_t   data;
    {
        free(data);
    }
```

Whenever an object is freed, all the "key data" objects are scanned. If "remove procedures" are associated with them, they are called with the key data as the parameter. In this case, `free_data()` is called, which frees the data associated with that particular key. The "remove procedure" is called only after the object has been completely destroyed; therefore, there should be no attempt to access the destroyed object in the procedure.

If you need to assign a new key to the *same* key data of an object, the old key data is automatically freed by XView by calling the remove procedure (if it exists). If you wish to delete a key without having to assign a new key, then you can call:

```
xv_set(object, XV_KEY_DATA_REMOVE, key, NULL);
```

7.19.2 Using PANEL_REPAINT_PROC

The `PANEL` package provides an property for installing a repaint routine. Warning: use of the repaint routine allows you to generate a **Non-OPEN LOOK** user interface.

Example 7-7 demonstrates how a repaint routine can be installed on a panel. This repaint routine draws a gray background on the panel behind any existing panel items.

Example 7-7. The panel_repaint.c program

```
/*
 * panel_repaint.c -- repaint a panel background without disturbing
 * the repainting of panel items.
 */
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/svrimage.h>
#include <X11/Xlib.h>
#include <X11/X.h>
#include <X11/bitmaps/gray1>
```

Example 7-7. The panel_repaint.c program (continued)

```
#define PANEL_GC_KEY    101  /* any arbitrary number */

main(argc, argv)
int argc;
char *argv[ ];
{
    Display     *display;
    Frame        frame;
    Panel        panel;
    int          quit();
    void         panel_repaint();
    XGCValues    gcvalues;
    Server_image grey;

    Mask        gcmask = 0L;
    GC          gc;

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

    frame = (Frame)xv_create(XV_NULL, FRAME, NULL);
    panel = (Panel)xv_create(frame, PANEL,
        PANEL_REPAINT_PROC,     panel_repaint,
        NULL);

    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Quit",
        PANEL_NOTIFY_PROC,      quit,
        PANEL_CLIENT_DATA,      frame,
        NULL);

    window_fit(frame);

    grey = (Server_image)xv_create(NULL, SERVER_IMAGE,
        XV_WIDTH,               gray1_width,
        XV_HEIGHT,              gray1_height,
        SERVER_IMAGE_DEPTH,     1, /* clarify for completeness*/
        SERVER_IMAGE_BITS,      gray1_bits,
        NULL);

    display = (Display *)xv_get(panel, XV_DISPLAY);
    gcvalues.stipple = (Pixmap) xv_get(grey, XV_XID);
    gcvalues.fill_style = FillOpaqueStippled;
    gcvalues.plane_mask = 1L;
    gcvalues.graphics_exposures = False;
    gcvalues.foreground = BlackPixel(display, DefaultScreen(display));
    gcvalues.background = WhitePixel(display, DefaultScreen(display));
    gcmask = GCStipple | GCFillStyle | GCPlaneMask |
        GCGraphicsExposures | GCForeground | GCBackground;
    gc = XCreateGC(display, xv_get(panel, XV_XID), gcmask, &gcvalues);

    /* attach the GC to the panel for use by the repaint proc above */
    xv_set(panel, XV_KEY_DATA, PANEL_GC_KEY, gc, NULL);

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

Example 7-7. The panel_repaint.c program (continued)

```
/*
 * repaint procedure for the panel paints a gray pattern over the
 * entire panel.  Use the GC attached to the panel via XV_KEY_DATA.
 */
void
panel_repaint(panel, pw, p_area)
Panel      panel;
Xv_Window  pw;
Rectlist   p_area;
{
    /* get the GC attached to the panel in main() */
    GC gc = (GC)xv_get(panel, XV_KEY_DATA, PANEL_GC_KEY);

    /* call XFillRectangle on the entire size of the panel window */
    XFillRectangle(xv_get(panel, XV_DISPLAY), xv_get(pw, XV_XID), gc,
        0, 0, xv_get(pw, XV_WIDTH), xv_get(pw, XV_HEIGHT));

    /* Note this repaints the entire panel.  It is best to  */
    /* repaint just the rectangles passed in p_area         */

}

quit(item)
Panel_item item;
{
    Frame frame = (Frame)xv_get(item, PANEL_CLIENT_DATA);
    xv_destroy_safe(frame);
}
```

The output produced by this program is shown in Figure 7-21.

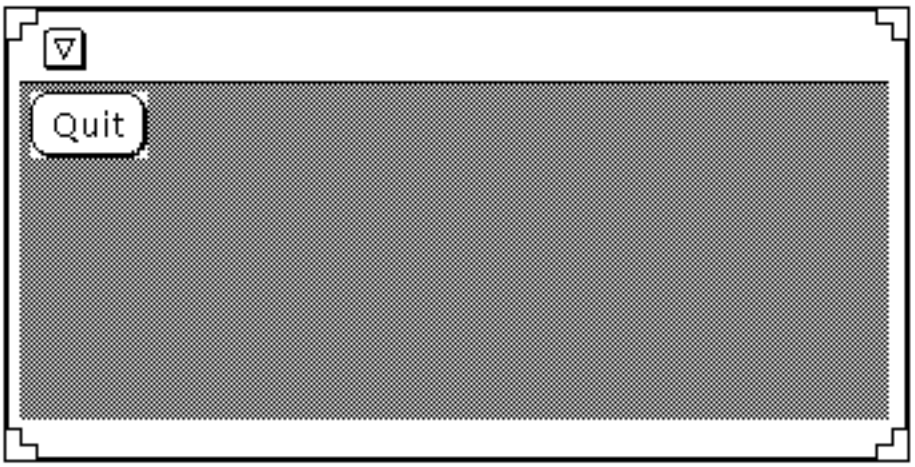

Figure 7-21. Panel with gray background

The PANEL package does not retain its paint windows by default, so the repaint routine may be called more frequently than one might expect. Therefore, when the panel is created, the attribute WIN_RETAINED may be set to TRUE; otherwise, the routine should try to be as computationally cheap as possible to maintain good performance. It is not recommended that you retain the panel's window unless you have provided a repaint routine that might utilize graphics expensively. Typically, you will set the background to a solid color, render a pattern, or display an image.

If a panel item is added, deleted or moved, then the repaint routine is called regardless of whether or not the panel's window is retained.

7.19.3 Painting Panel Items

To repaint either an individual item or an entire panel, use:

```
panel_paint(panel_object, paint_behavior)
    Panel_item    panel_object;
    Panel_setting paint_behavior;
```

The panel_object may be a panel item or a panel itself. If it is a panel, the items within the panel are repainted one by one. The argument paint_behavior is either PANEL_CLEAR, which causes the rectangle occupied by the panel or item to be cleared prior to repainting, or PANEL_NO_CLEAR, which causes repainting to be done without any prior clearing. This setting will override the default paint behavior set in the panel item's PANEL_PAINT attribute.

7.19.4 Panel Event Handling

This section describes how the PANEL package handles events. If you require a behavior not provided by default, you can write your own event handling procedure for either an individual item or the panel as a whole. The default behavior for handling panel events conforms to OPEN LOOK and should be sufficient for most users. This section is intended *only* for expert users.

WARNING

Changing the default PANEL package event handling behavior allows you to create applications that are *not* OPEN LOOK-compliant.

The default event handling mechanism for panels processes events for all the panel items in a uniform way. A single routine reads the events, updates an internal state machine, and maps the event to an *action* to be taken by the item. Actions fall into two categories: *previewing* and *accepting*. The previewing action gives the user visual feedback indicating what will happen when the mouse button is released. The accepting action causes the item's value to be changed and/or its notify procedure to be called, with the event passed as an argument.

The default event-to-action mapping is given in Table 7-3.

Table 7-3. Default Event to Action Mapping

Event	Action
SELECT button down or drag with SELECT button down.	Begin previewing.
Drag with SELECT button down.	Update previewing.
Drag out of item rectangle with SELECT button down.	Cancel preview.
SELECT button up	Accept.
MENU button down	Display menu & accept user's selection.
Keystroke	Accept keystroke if text item.

What actually happens when an item is told to perform one of the above actions depends on the type of item. For example, when asked to begin previewing, a button item inverts its label, a message item does nothing, a slider item redraws the shaded area of its slider bar, etc.

ASCII events and some `action` events (described in Chapter 6, *Handling Input*) are automatically redirected towards the item with keyboard focus. Since only one item at a time may receive keyboard events, if there is more than one item in the panel that can receive keyboard input, the one that is receiving the keyboard events has a solid location cursor (a small or large solid triangle, also referred to as a caret). You may use `PANEL_CARET_ITEM` with `xv_set()` or `xv_get()` to set or get the item that currently has the keyboard focus.

Handling events by the application in panels is a task best avoided since the panel does this automatically. But there are certainly situations where the application might like to supervise or handle events itself. In such situations, there are several methods available for event handling. You can use:

- `notify_interpose_event_func()`
- `PANEL_BACKGROUND_PROC`
- `PANEL_EVENT_PROC`

For normal panels, each of these methods should be used on the panel itself. However, for the `SCROLLABLE_PANEL`, the event handler must be set on the panel's paint window(s) exactly as is done for canvases (using the attribute `WIN_EVENT_PROC`).

Each of these methods works somewhat differently from one another, but they all have one thing in common: they are notified when events happen in panels.

7.19.5 Using an Interpose Function

The Notifier's interpose functions may be installed on panels just as they are for any other window-based package. This is the recommended method for special event handling for panels and for panel items, since it allows you to interfere with the normal event processing for the destination panel (or panel item). However, events can continue to be dispatched to the panel (panel item) through the use of the `notify_next_event_func()`. Refer to Chapter 20, *The Notifier*, for details on interposition.

7.19.6 Using PANEL_BACKGROUND_PROC

The `PANEL_BACKGROUND_PROC` is similar to the `WIN_EVENT_PROC` except that the notification routine is only notified of events that do not happen in, or are redirected to, any panel items. The application would want to know about events that are not sent to panel items:

```
extern void my_event_proc();

panel = (Panel)xv_create(frame, PANEL,
    PANEL_BACKGROUND_PROC,   my_event_proc,
    NULL);
```

The parameters to the routine for `PANEL_BACKGROUND_PROC` are:

```
void
my_event_proc(panel, event)
    Panel  panel;
    Event *event;
```

The `PANEL_BACKGROUND_PROC` does not, by default, get keyboard events passed to it. Therefore, rather than trying to set this mask explicitly in the panel's window, the attribute `PANEL_ACCEPT_KEYSTROKE` can be set to `TRUE`. With this attribute set, ASCII events and function-key events are passed to the routine, provided there are no panel items that accept keyboard input. If there are such items, those panel items will continue to get keyboard events regardless of the attribute `PANEL_ACCEPT_KEYSTROKE`. If you wish to get keyboard events instead of the panel items that consume those events, you should use an event interposing function discussed in Chapter 20.

7.19.7 Using PANEL_EVENT_PROC

Just as `PANEL_BACKGROUND_PROC` specifies a routine to handle events that happen outside of panel items, you can also get events that happen only *within* panel items using `PANEL_EVENT_PROC`.

```
xv_set(panel, PANEL_EVENT_PROC, my_event_proc, NULL);
```

Using this routine causes the default event handling for that item to be ignored in favor of the new event procedure. In other words, this routine *does* interfere with the normal event processing for panel items, and the panel item's callback routine is no longer automatically called by the `PANEL` package.

Applications can get the event handler routine for a panel item with the following call:

```
Panel_item item;
event_proc = (void (*)())xv_get(item, PANEL_EVENT_PROC);
```

You may not assume that the default event handler for any panel item is `panel_default_handle_event()`, as in previous XView versions. You must `xv_get()` the item's `PANEL_EVENT_PROC`. The event handler has the following parameters:

```
void
panel_item_event_proc(item, event)
    Panel_item   item;
    Event       *event;
```

7.19.8 Event Handling Example

The program *item_move.c* (a longer program listed in Appendix F, *Example Programs*) demonstrates how events can be handled in panels. The program allows you to create, destroy, and move around three different types of panel items. Two panels are displayed (see Figure 7-22) The control panel contains two panel items: a text item to type in panel item names and a choice item providing the different item types that may be created. The destination panel is where newly created panel items are placed. It is boxed. After items are created, you may move them around the destination panel using the MENU mouse button. Moving the item off the destination panel deletes the item.

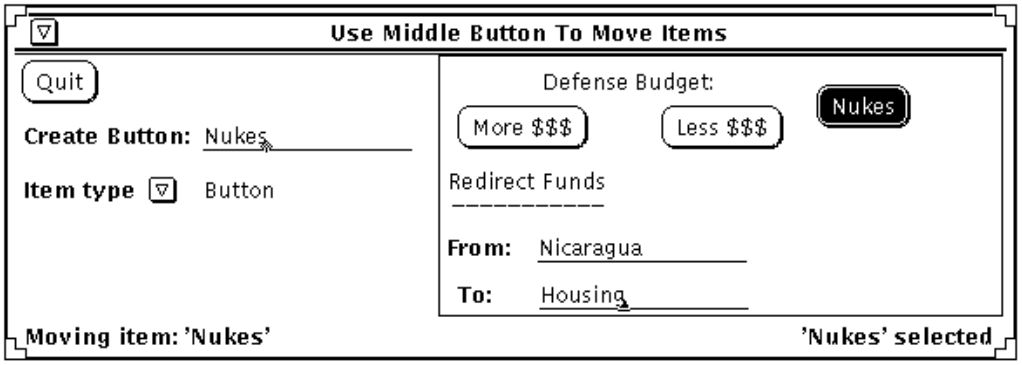

Figure 7-22. Output of item_move.c in use

The code for *item_move.c* uses the interpose function, `notify_interpose_event_func()`, to handle events within the destination panel. This interferes with the normal event processing for the destination panel; however, events continue to be dispatched to the panel items through the use of the `notify_next_event_func()`. Since the event function is only interested in MENU button events that occur on the destination panel's panel items, it might seem appropriate to use `PANEL_EVENT_PROC` since it is designed to notify the routine only when panel items receive events. However, this method would not work for this application because when the mouse button is dragged around the

frame to move the item, the dragging events might move outside of the button and when that occurred the event callback would not be called.

7.20 Panel Package Summary

Table 7-4 lists the procedures and macros for the PANEL package. Table 7-5 lists the attributes for the PANEL package. This information is fully described in the *XView Reference Manual*.

Table 7-4. Panel Procedures and Macros

Procedures and Macros	
panel_advance_caret()	PANEL_CHECK_BOX
panel_backup_caret()	PANEL_CHOICE_STACK
panel_paint()	PANEL_EACH_ITEM()
panel_text_notify()	PANEL_END_EACH()
	PANEL_TOGGLE

Table 7-5. Panel Package Attributes

Panel Attributes	
PANEL_ACCEPT_KEYSTROKE	PANEL_LAYOUT
PANEL_BACKGROUND_PROC	PANEL_LINE_BREAK_ACTION
PANEL_BUSY	PANEL_LIST_CLIENT_DATA
PANEL_CARET_ITEM	PANEL_LIST_CLIENT_DATAS
PANEL_CHILD_CARET_ITEM	PANEL_LIST_DELETE
PANEL_CHOICE_COLOR	PANEL_LIST_DELETE_ROWS
PANEL_CHOICE_IMAGE	PANEL_LIST_DELETE_SELECTED_ROWS
PANEL_CHOICE_IMAGES	PANEL_LIST_DISPLAY_ROWS
PANEL_CHOICE_NCOLS	PANEL_LIST_FIRST_SELECTED
PANEL_CHOICE_NROWS	PANEL_LIST_FONT
PANEL_CHOICE_RECT	PANEL_LIST_FONTS
PANEL_CHOICE_STRING	PANEL_LIST_GLYPH
PANEL_CHOICE_STRINGS	PANEL_LIST_GLYPHS
PANEL_CHOOSE_NONE	PANEL_LIST_INSERT
PANEL_CHOOSE_ONE	PANEL_LIST_INSERT_DUPLICATE
PANEL_CLIENT_DATA	PANEL_LIST_INSERT_GLYPHS
PANEL_CURRENT_ITEM	PANEL_LIST_INSERT_STRINGS
PANEL_DEFAULT_ITEM	PANEL_LIST_MODE
PANEL_DEFAULT_VALUE	PANEL_LIST_NEXT_SELECTED
PANEL_DIRECTION	PANEL_LIST_NROWS
PANEL_DISPLAY_LEVEL	PANEL_LIST_ROW_HEIGHT
PANEL_DISPLAY_ROWS	PANEL_LIST_SCROLLBAR
PANEL_DROP_BUSY_GLYPH	PANEL_LIST_SELECT

Table 7-5. Panel Package Attributes (continued)

Panel Attributes

PANEL_DROP_DND	PANEL_LIST_SELECTED
PANEL_DROP_FULL	PANEL_LIST_SORT
PANEL_DROP_GLYPH	PANEL_LIST_STRING
PANEL_DROP_HEIGHT	PANEL_LIST_STRINGS
PANEL_DROP_SEL_REQ	PANEL_LIST_TITLE
PANEL_DROP_SITE_DEFAULT	PANEL_LIST_WIDTH
PANEL_DROP_WIDTH	PANEL_MASK_CHAR
PANEL_EVENT_PROC	PANEL_MAX_TICK_STRING
PANEL_EXTRA_PAINT_HEIGHT	PANEL_MAX_VALUE
PANEL_EXTRA_PAINT_WIDTH	PANEL_MAX_VALUE_STRING
PANEL_FEEDBACK	PANEL_MIN_TICK_STRING
PANEL_FIRST_ITEM	PANEL_MIN_VALUE
PANEL_FIRST_PAINT_WINDOW	PANEL_MIN_VALUE_STRING
PANEL_FOCUS_PW	PANEL_NCHOICES
PANEL_GAUGE_WIDTH	PANEL_NEXT_COL
PANEL_GINFO	PANEL_NEXT_ITEM
PANEL_INACTIVE	PANEL_NEXT_ROW
PANEL_ITEM_CLASS	PANEL_NO_REDISPLAY_ITEM
PANEL_ITEM_COLOR	PANEL_NOTIFY_LEVEL
PANEL_ITEM_CREATED	PANEL_NOTIFY_PROC
PANEL_ITEM_DEAF	PANEL_NOTIFY_STATUS
PANEL_ITEM_LABEL_RECT	PANEL_NOTIFY_STRING
PANEL_ITEM_MENU	PANEL_OPS_VECTOR
PANEL_ITEM_NTH_WINDOW	PANEL_PAINT
PANEL_ITEM_NWINDOWS	PANEL_PRIMARY_FOCUS_ITEM
PANEL_ITEM_RECT	PANEL_READ_ONLY
PANEL_ITEM_VALUE_RECT	PANEL_REPAINT_PROC
PANEL_ITEM_WANTS_ADJUST	PANEL_SHOW_RANGE
PANEL_ITEM_WANTS_ISO	PANEL_SHOW_VALUE
PANEL_ITEM_X	PANEL_SLIDER_END_BOXES
PANEL_ITEM_X_GAP	PANEL_SLIDER_WIDTH
PANEL_ITEM_X_POSITION	PANEL_STATUS
PANEL_ITEM_Y	PANEL_TEXT_SELECT_LINE
PANEL_ITEM_Y_GAP	PANEL_TICKS
PANEL_ITEM_Y_POSITION	PANEL_TOGGLE_VALUE
PANEL_JUMP_DELTA	PANEL_VALUE
PANEL_LABEL_BOLD	PANEL_VALUE_DISPLAY_LENGTH
PANEL_LABEL_FONT	PANEL_VALUE_DISPLAY_WIDTH
PANEL_LABEL_IMAGE	PANEL_VALUE_FONT
PANEL_LABEL_STRING	PANEL_VALUE_STORED_LENGTH
PANEL_LABEL_WIDTH	PANEL_VALUE_UNDERLINED
PANEL_LABEL_X	PANEL_VALUE_X
PANEL_LABEL_Y	PANEL_VALUE_Y

Table 7-6. New and Changed Panel Package Attributes (Version 3.2)

PANEL_LIST_INACTIVE	PANEL_LIST_MASK_GLYPHS
PANEL_LIST_DELETE_INACTIVE_ROWS	PANEL_LIST_ROW_VALUES
PANEL_LIST_DO_DBL_CLICK	PANEL_LIST_EXTENSION_DATA
PANEL_LIST_MASK_GLYPH	PANEL_LIST_EXTENSION_DATAS