11
Menus

Menus play an important role in an application's user interface. An OPEN LOOK menu may display text or graphics. Menus may be attached to most XView objects such as *menu buttons*, *scrollbars* or *text subwindows*, or they may exist independently from objects and be displayed on demand.

The user may cause a menu to be pinned up by selecting an optional *pushpin* in the pop-up menu. When this happens, the menu is taken down and a corresponding command frame is put up at the same location. Panel items in the pinup window correspond to the menu items in the menu. Once a menu has been pinned up, the user continues to interact with it just as if the menu were popped up each time. Menus that are used frequently are good candidates for having pushpins so the user does not have to repeat the sequence of redisplaying the menu to make selections.

OPEN LOOK requires that menus have titles. Menus or submenus that originate from *menu buttons* or *pullright* items do not need to have titles, since the name of the menu button or menu item acts as the title.

Fonts may not be specified in either menu items or menu titles; menu items follow the same constraints outlined for *panel buttons*. However, if text is not used, then menu items may contain graphic images, in which case, the font is of no concern. That is, you could specify a `Server_image` that has a string rendered in a particular font.

11.1 Menu Types

There are three different types of menus: *pop-up*, *pulldown*, and *pullright* menus. The general term *pop-up menu* may describes all three types in certain contexts since menus are *popped up*. However, pulldown and pullright menus have distinct characteristics that make them unique.

11.1.1 Pop-up Menus

Pop-up menus are displayed when the user selects the MENU mouse button over XView objects such as *scrollbars* or *text subwindows*. An OPEN LOOK window manager also utilizes pop-up menus in the root window and from base frame title bars. XView objects handle the display of menus automatically. Applications may wish to track `ACTION_MENU` events in objects such as canvases and display their own pop-up menus. Figure 11-1, from the *OPEN LOOK GUI Specification Guide*, shows a *Window* menu-generated from the title bar of an OPEN LOOK base frame.

Figure 11-1. The Window menu

11.1.2 Pulldown Menus

Pulldown menus are attached to *menu buttons*. Menu buttons have a set of choices associated with them that the user can access only via the pulldown menu. When the user presses the MENU mouse button over a menu button, the choices are displayed in the form of a pulldown menu. If the menu button is selected using the SELECT button, the default menu item is selected. See Chapter 7, *Panels*, for details on creating menu buttons. Figure 11-2, from the *OPEN LOOK GUI Specification Guide*, shows sample pulldown menus activated from a menu button.

11.1.3 Pullright Menus

OPEN LOOK provides for items in the menu to have *pullright menus* associated with them. Also called *cascading menus*, these menus are activated from the user dragging the MENU mouse button to the right of a menu item that has an arrow pointing to the right. The cascading menu that results is a pop-up menu that can also have menu items with pullrights attached. Figure 11-3, from the *OPEN LOOK GUI Specification Guide*, shows a pullright menu originating from a menu item in a pulldown menu.

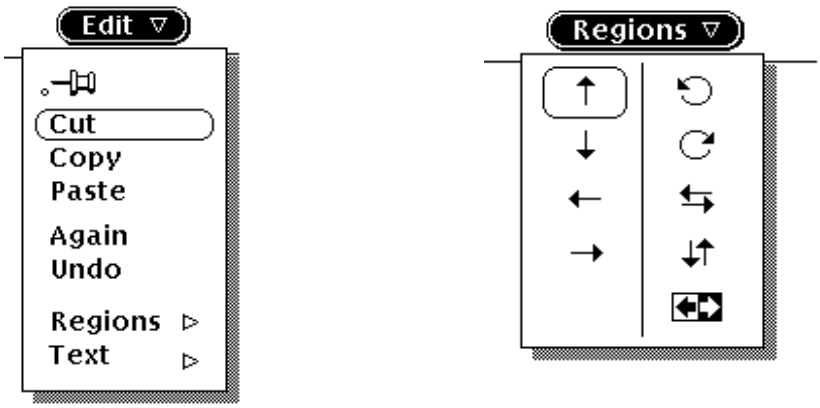

Figure 11-2. Menu buttons each with a pulldown menu

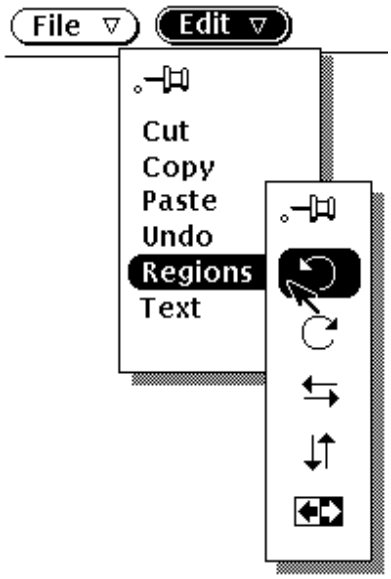

Figure 11-3. Pushpins in a menu and a submenu

11.2 Menu Items

In addition to the menu types, there are different types of menu items: *choice, exclusive*, and *nonexclusive*. The different menu item types may be associated with each type of menu.

Each menu has a *default selection* associated with it. This item is displayed uniquely from other menu items and designates a default action to take if the user wants to select the menu without displaying it (see *pulldown menus* below). Typically, the 0th item in the menu is the default, but that may be changed either by the application or by the user.

11.2.1 Choice Items

The *choice* item is the default menu item type used when a menu is created. The default selection in a menu has a ring around it. When a pop-up menu is displayed, it is positioned so that the mouse is pointing at the default item. Choice menu items may have pullright menus associated with them, in which case there is a pullright arrow at the right side of the item. If the selection of a menu item brings up a dialog box (command frame), then the label for the menu item typically ends in ellipses (. . .).

11.2.2 Exclusive Items

When a choice item is selected, an action is taken and the menu forgets about it. Exclusive menu items retain the fact that they are selected even after the menu has popped down. If the user selects a new item, the new item is remembered. Because this is an exclusive menu, only one choice may be selected at a time. The *default* item is indicated by a double-lined box around the item. Figure 11-4 is from the *OPEN LOOK GUI Specification Guide*.

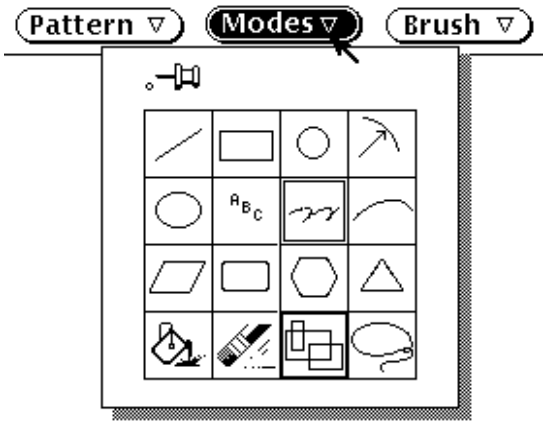

Figure 11-4. Exclusive settings on a menu

When exclusive settings are used on menus, the current choice has a bold border when the pointer is not on a menu choice. When the user drags the pointer onto other settings, the bold border follows the pointer. Exclusive choice menus may not have items with pullright menus.

11.2.3 Nonexclusive Items

Also called *toggle items*, menus that have toggle items support multiple choices from the menu to be selected at the same time. That is, the user may *toggle* whether a particular choice is selected. This action has no affect on the other menu items. Figure 11-5 shows an example of a menu that has items and a submenu that has nonexclusive settings.

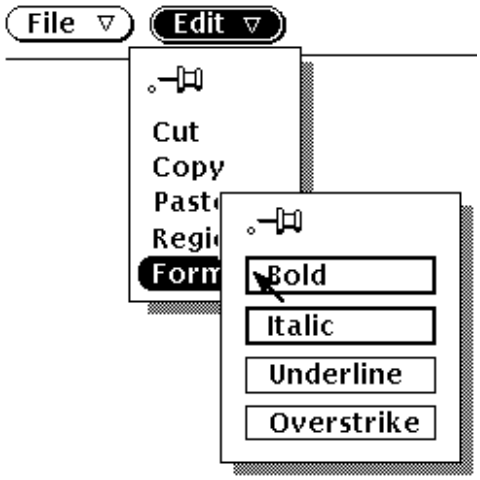

Figure 11-5. Nonexclusive settings on a submenu

In this figure, the chosen settings on the submenu are Bold and Italic. The choices not selected are Underline and Overstrike.

11.3 Creating Menus

The header file for the `MENU` package is *<xview/openmenu.h>*, but the file is already included by *<xview/xview.h>*. Another name for the `MENU` package is `MENU_COMMAND_MENU`. The basic menu is created using `xv_create()`:

```
Menu menu;

menu = (Menu)xv_create(server, MENU, NULL);
```

Menus (and sometimes menu items) are discrete objects that may have *delayed binding*. That is, they may be created independently from any XView object and attached later using

attributes specific to that object, such as the way that PANEL_ITEM_MENU can be used to attach an existing menu to a *menu button*. However, the association between menus and the items they are attached to does not imply "ownership."

The owner of a menu is a server object. By default, (if NULL is specified as the owner) the default server is used. Menus may be used only on the server specified; they may not be shared across different servers. Thus, the menu owner is only a concern for applications that spread across multiple servers. See Chapter 15, *Nonvisual Objects*, for details on opening a connection to different servers.

The *parent* of a menu, however, may be a pullright item from another menu. See Section 11.8, "Pullright Menus," later in this chapter.

Figure 11-6 shows the class hierarchy for the a menu object.

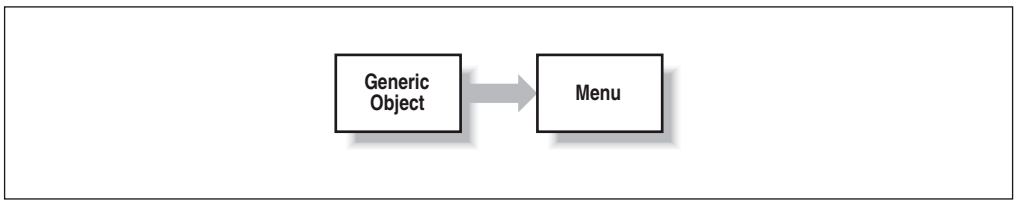

Figure 11-6. Menu class hierarchy

Exclusive menus are created using the MENU_CHOICE_MENU package, as in the following example:

```
Menu menu;

menu = (Menu)xv_create(NULL, MENU_CHOICE_MENU,
    MENU_STRINGS,  "choice1", "choice2", "choice3", NULL,
    NULL);
```

Nonexclusive menus are created using the MENU_TOGGLE_MENU package, as in the following example:

```
Menu menu;
Server_image image1, image2, image3;

menu = (Menu)xv_create(NULL, MENU_TOGGLE_MENU,
    MENU_IMAGES,  image1, image2, image3, NULL,
    NULL);
```

11.4 Displaying Menus

Menus are displayed (popped up) using the function `menu_show()`.* It displays the specified menu and immediately returns. The function takes the form:

```
void
menu_show(menu, window, event, NULL);
    Menu          menu;
    Xv_Window     window;
    Event        *event;
```

The `menu` is a menu created from `xv_create()` or a menu extracted from an existing XView object (such as a button menu). The X window associated with the menu calls `XGrabPointer()` to grab the server's mouse events. The pointer grab stays in effect until the user releases the MENU mouse button (e.g., the `ACTION_MENU` action with `event_is_up()` being `TRUE`). This is independent of the event that caused the menu to be displayed. Releasing the MENU button results in the user having either made a selection, *not* made a selection or pinned up the menu (provided that the menu has a pushpin).

The `window` attribute defines the window where the menu appears. The `event` parameter contains the event that caused the decision to display the menu. The most common use for it is to extract the *x,y* coordinate pair so as to remember the location of the pointer at the time the menu was displayed. This event structure can be retrieved later by calling:

```
Event *event = (Event *)xv_get(menu, MENU_FIRST_EVENT);
```

Similarly, when the user releases the MENU button, this event can be retrieved using the `MENU_LAST_EVENT` attribute.

The last parameter to `menu_show()` must be `NULL`. It actually represents a list of attribute-value pairs but is used internally by other XView packages that utilize menus.

Menus can also be created at run time by procedures that are called whenever a menu is needed. This is covered in Section 11.9, "Menu-generating Procedures."

The routine `MENU_DONE_PROC` is called whenever a pop-up menu has been taken down (after a menu item has been selected), pinned up, or simply dismissed without a selection being made. This overrides the default action of setting `XV_SHOW` to `FALSE`, so this responsibility lies with the `MENU_DONE_PROC` routine.

*This function is called internally by other XView objects such as scrollbars, menu buttons, and text subwindows to display menus associated with them.

11.5 A Simple Program

Given the information provided so far, we can demonstrate how to pop up a menu. Example 11-1 shows how the canvas object tracks pointer events and calls `menu_show()` when the `ACTION_MENU` event occurs.

Example 11-1. The simple_menu.c program

```
/*
 * simple_menu.c -
 * Demonstrate the use of an XView menu in a canvas subwindow.
 * A Menu is brought up with the MENU mouse button.  The choices
 * in the menu toggle the display of the scrollbar next to the canvas.
 */
#include <xview/xview.h>
#include <xview/canvas.h>
#include <xview/scrollbar.h>

#define SCROLLBAR_KEY   100
#define MENU_KEY        200

main(argc,argv)
int     argc;
char    *argv[ ];
{
    Frame       frame;
    Canvas      canvas;
    Scrollbar   scrollbar;
    Menu        menu;
    void        menu_notify_proc(), pw_event_proc();

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


    /*
     *  Create a frame, canvas and menu.
     *  A canvas receives input in its canvas_paint_window().
     */
    frame = (Frame)xv_create(NULL, FRAME,
        FRAME_LABEL,    argv[0],
        NULL);
    canvas = (Canvas)xv_create(frame, CANVAS,
        XV_WIDTH,       300,
        XV_HEIGHT,      200,
        NULL);
    scrollbar = (Scrollbar)xv_create(canvas, SCROLLBAR,
        SCROLLBAR_DIRECTION,    SCROLLBAR_VERTICAL,
        NULL);

    menu = (Menu)xv_create(NULL, MENU,
        MENU_TITLE_ITEM,        "Scrollbar",
        MENU_STRINGS,           "On", "Off", NULL,
        MENU_NOTIFY_PROC,       menu_notify_proc,
        XV_KEY_DATA,            SCROLLBAR_KEY, scrollbar,
        NULL);
```

Example 11-1. The simple_menu.c program (continued)

```
    xv_set(canvas_paint_window(canvas),
        WIN_EVENT_PROC,         pw_event_proc,
        XV_KEY_DATA,            MENU_KEY, menu,
        NULL);

    window_fit(frame);
    window_main_loop(frame);
}

/*
 * menu_notify_proc - toggle the display of the scrollbar
 * based on which menu item was chosen.
 */
void
menu_notify_proc(menu, menu_item)
Menu menu;
Menu_item menu_item;
{
    char *menu_choice = (char *)xv_get(menu_item, MENU_STRING);
    int show_it = !strcmp(menu_choice, "On");

    xv_set(xv_get(menu, XV_KEY_DATA, SCROLLBAR_KEY),
        XV_SHOW,        show_it,
        NULL);
}

/*
 * Call menu_show() to display menu.
 */
void
pw_event_proc(canvas_pw, event)
Xv_Window       canvas_pw;
Event *event;
{
    if (event_action(event) == ACTION_MENU && event_is_down(event)) {


        Menu menu = (Menu)xv_get(canvas_pw, XV_KEY_DATA, MENU_KEY);
        menu_show(menu, canvas_pw, event, NULL);
    }
}
```

In Example 11-1 above, *simple_menu.c* shows the simplest and most common method for creating and using pop-up menus. The menu, menu items, and callback routine are all created at the same time using the call:

```
menu = (Menu)xv_create(NULL, MENU,
    MENU_TITLE_ITEM,        "Scrollbar",
    MENU_STRINGS,           "On", "Off", NULL,
    MENU_NOTIFY_PROC,       menu_notify_proc,
    NULL);
```

Figure 11-7 shows the result of running *simple_menu.c* and selecting the menu.

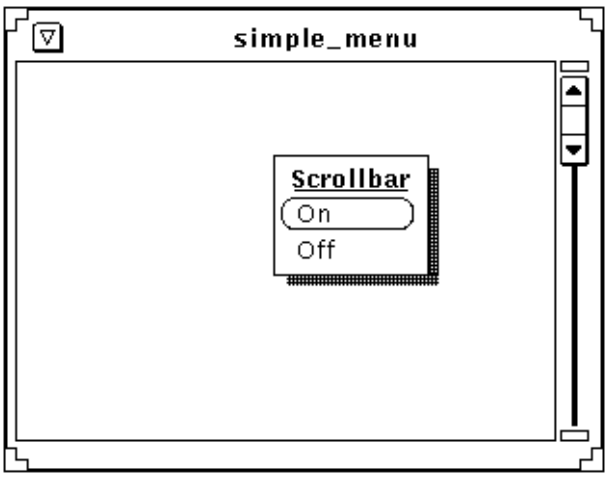

Figure 11-7. Output of simple_menu.c when the menu is popped up

Since this menu is not attached to a menu button panel item and is not a pullright menu of another menu item, a title bar is added using `MENU_TITLE_ITEM`. When a menu has a title, the 0th menu item is the menu title. In the example above, the 0th menu item is the title item labeled "Scrollbar." The first menu item is the first item created after the title. Above, the first menu item is the item labeled "On" and the second menu item is the item labeled "Off." When there is no title, the first menu item is placed in position zero. Add a title bar using `MENU_TITLE_ITEM` *only* if the menu is a pop-up menu. Using titles on menus originating from menu buttons is *not* OPEN LOOK-compliant.

The attribute `MENU_STRINGS` takes a list of strings and creates a menu item for each string.

NOTE

> The `MENU` package, in contrast to the `PANEL` package, does not save strings which you pass in as the menu item's label (string). You should either pass a constant string, as in the example above, or static storage that you have dynamically allocated (e.g., `malloc()`).

`menu_notify_proc()` is a routine that is called whenever any of the menu items are selected. The routine is passed a handle to the menu and the menu item selected. The canvas serves no other purpose than to capture events—the event callback routine for the canvas determines if the user generated the `ACTION_MENU` event and, if so, calls `menu_show()`.

The use of `XV_KEY_DATA` is used to associate one object with another. In this case, we associate the scrollbar with the menu so when the menu callback routine is called, the scrollbar can be retrieved easily. The menu, on the other hand, is associated with the canvas's paint window since that window is going to get the event that pops up the menu.

The use of XV_KEY_DATA removes the need for the menu and the scrollbar to be global variables. It also clarifies the association between the objects that would otherwise not be as clear. Other examples of XV_KEY_DATA, including a complete discussion on usage, can be found in Chapter 7, *Panels*, and Chapter 3, *Creating XView Applications*.

11.6 Creating Menu Items

As noted, the use of MENU_STRINGS results in the creation of menu items *in-line*; that is, they are created automatically by the MENU package during the creation of the menu object. Other methods for creating menu items in-line include using the attributes, MENU_ITEM and MENU_ACTION_ITEM.*

MENU_ACTION_ITEM is used as a shortcut for separately specifying a label and a callback routine:

```
menu = (Menu)xv_create(NULL, MENU,
    MENU_ACTION_ITEM,  "item1",  callback1,
    MENU_ACTION_ITEM,  "item2",  callback2,
    ...
    NULL);
```

Two other methods for creating menu items are to:

- Use a separate call to xv_create() with the MENUITEM package.
- Provide a menu-generation routine.

The following subsections discuss the use of MENU_ITEM to create menu items in-line, MENUITEM to create separate menu items, and MENU_GEN_PROC to specify a routine that creates menus.

11.6.1 Using MENU_ITEM

Using the attribute MENU_ITEM indicates that a new menu item is to be created and appended to the existing menu. Use of this attribute means that the menu item is created in-line so a separate call to xv_create() is not necessary. The form of the value portion of the attribute is a NULL-terminated list of menu-item, attribute-value pairs:

```
extern void on_notify_proc(), off_notify_proc();
menu = (Menu)xv_create(NULL, MENU,
    MENU_TITLE_ITEM,      "Scrollbar",
    MENU_ITEM,
        MENU_STRING,      "On",
        MENU_NOTIFY_PROC, on_notify_proc,
        NULL,
    MENU_ITEM,
        MENU_STRING,      "Off",
        MENU_NOTIFY_PROC, off_notify_proc,
        NULL,
```

*The attributes MENU_IMAGES and MENU_ACTION_IMAGE are just like MENU_STRINGS and MENU_ACTION_ITEM except that Server_images are used as labels rather than text.

```
    NULL);
```

The code fragment shown above creates a menu with the same two menu items as shown in the previous example, except that the menu items are created more directly by the use of MENU_ITEM. Here we can specify item-specific attributes rather than accept all the defaults for the menu. In this case, we set a different notification routine for each menu item.*

If you have a Server_image to display rather than a string, you can replace MENU_STRING above with MENU_IMAGE and specify a Server_image rather than a string.

11.6.2 Using MENU_ACTION_ITEM

Rather than specifying menu item creation using a separate attribute-value list, the attribute MENU_ACTION_ITEM can be used as a shortcut, as shown in the example below:

```
menu = (Menu)xv_create(NULL, MENU,
    MENU_TITLE_ITEM,      "Scrollbar",
    MENU_ACTION_ITEM,     "On",    on_notify_proc,
    MENU_ACTION_ITEM,     "Off",   off_notify_proc,
    NULL);
```

The attribute MENU_ACTION_IMAGE, with a Server_image as its value, may be used interchangeably with MENU_ACTION_ITEM and its string value.

11.6.3 Using MENUITEM

The MENUITEM package allows you to create separate menu items using separate calls to xv_create(). The attributes used are menu item-specific attributes—the same as those that are used in the MENU_ITEM attribute above.

```
Menu_item   on, off;

on = (Menu_item)xv_create(NULL, MENUITEM,
    MENU_STRING,        "On",
    MENU_NOTIFY_PROC,   on_notify_proc,
    NULL);

off = (Menu_item)xv_create(NULL, MENUITEM,
    MENU_STRING,        "Off",
    MENU_NOTIFY_PROC,   off_notify_proc,
    NULL);

xv_set(menu,
    MENU_APPEND_ITEM,   on,
    MENU_APPEND_ITEM,   off,
    NULL);
```

These menu items are not created *in-line*; they are created independently using separate calls to xv_create(). They must therefore be added to the menu independently. In this case,

*Notification (callback) routines are discussed in Section 11.13, “Notification Procedures.”

they are added using MENU_APPEND_ITEM (see the next section for more information).

11.7 Adding Menu Items

There are several methods for adding separately created menu items to menus. For a list of menu item attributes, see the package summary at the end of this chapter. These attributes can be used when you are using xv_set() on a menu.

The code fragment below demonstrates the use of MENU_APPEND_ITEM. The menu items are created independently of the menu itself. They are added to the menu as they are created by using the attribute MENU_APPEND_ITEM:

```
char         *names[ ] = { "One", "Two", "Three", "Four", "Five" };
Menu         menu;
Menu_item    mi;
int          i;
void         my_notify_proc();

menu = (Menu)xv_create(NULL, MENU, NULL);

for (i = 0; i < 5; i++) {
    mi = (Menu_item)xv_create(NULL, MENUITEM,
        MENU_STRING,        names[i],
        MENU_NOTIFY_PROC,   my_notify_proc,
        MENU_RELEASE,
        NULL);
    xv_set(menu, MENU_APPEND_ITEM, mi, NULL);
}
```

This use of MENU_RELEASE is to indicate that the menu item is intended to be freed when the item's parent menu is destroyed. This attribute takes *no value*; specifying it is equivalent to specifying a TRUE value. Not specifying it implies FALSE. In-line menu items have this attribute set by default, but menu items that are not created in-line must set this attribute explicitly if you want them to be freed automatically. You do not want to set this attribute if you intend to use this menu item in more than one menu or if you want to reuse it later. See Section 11.17, "Destroying Menus."

11.8 Pullright Menus

A pullright menu is simply another menu that is attached to a menu item. Note that for a menu item to contain a pullright menu, the pullright menu must already have been created. This means that any menu group should be created from the bottom up. The attributes MENU_PULLRIGHT, MENU_PULLRIGHT_ITEM, and MENU_PULLRIGHT_IMAGE all allow a pullright menu to be attached to a menu item.

In the first case, MENU_PULLRIGHT can be assigned to a menu item to attach an existing menu to it, as shown below:

```
extern Server_image image1, image2, image3;
```

```
Menu image_menu, menu;
void image_notify_proc();

image_menu = (Menu)xv_create(NULL, MENU,
    MENU_IMAGES,       image1, image2, image3, NULL,
    MENU_NOTIFY_PROC,  image_notify_proc,
    NULL);

menu = (Menu)xv_create(NULL, MENU,
    MENU_ITEM,
        MENU_STRING,     "images",
        MENU_PULLRIGHT,  image_menu,
        NULL,
    NULL);
```

In the previous example, a menu of server images is created and is initialized to contain three images. Another menu is created that is initialized for one menu item, but that menu item has a pullright menu that is set to the `menu_images` menu.

The menu item created may also be initialized using the `MENU_PULLRIGHT_ITEM` attribute. This attribute takes two parameters as its *value*: a string and a menu. Therefore, the above code fragment could have been written:

```
menu = (Menu)xv_create(NULL, MENU,
    MENU_PULLRIGHT_ITEM, "images", image_menu,
    NULL);
```

Had the label for the pullright menu item been a `Server_image` rather than a string, the call would look like:

```
extern Server_image label_image;

menu = (Menu)xv_create(NULL, MENU,
    MENU_PULLRIGHT_IMAGE, label_image, image_menu,
    NULL);
```

In the code fragment below, we use another piece of code to demonstrate the same principle. This example demonstrates how a menu that represents font sizes may be set as the pullright menu for a list of fonts:

```
Menu       font_menu, size_menu;
Menu_item  mi;
int        i;
void       notify_font();
char       buf[4], *p;
...
size_menu = (Menu)xv_create(NULL, MENU,
    MENU_NOTIFY_PROC,   notify_size,
    NULL);
```

```
for (i = 8; i <= 20; i += 2) {
    sprintf(buf, "%d", i);
    p = strcpy(malloc(strlen(buf)+1), buf);
    mi = (Menu_item)xv_create(NULL, MENUITEM,
        MENU_STRING,      p,
        MENU_RELEASE,
        MENU_RELEASE_IMAGE,
        MENU_NOTIFY_PROC, notify_font,
        NULL);
    xv_set(size_menu, MENU_APPEND_ITEM, mi, NULL);
}

font_menu = (Menu)xv_create(NULL, MENU,
    MENU_TITLE_ITEM,     "Fonts",
    MENU_PULLRIGHT_ITEM, "courier",        size_menu,
    MENU_PULLRIGHT_ITEM, "boston",         size_menu,
    MENU_PULLRIGHT_ITEM, "times-roman",    size_menu,
    MENU_PULLRIGHT_ITEM, "lucidasans",     size_menu,
    MENU_PULLRIGHT_ITEM, "palatino-roman", size_menu,
    NULL);
```

Each item in the main menu (`font_menu`) has a pullright menu (`size_menu`) associated with it. In the `for` loop where the string for the menu item is assigned, the data is allocated using `malloc()` and `buf` is copied into the allocated data using `strcpy()`. We cannot use `buf` directly, because unlike panel items, the menu item string is not copied by the `MENU` package—we must pass in allocated data. Because of this, we also specify the attribute `MENU_RELEASE_IMAGE` so that when the item is destroyed, the allocated data will be freed. Also note that because we used `xv_create()` to create the menu item, we specify `MENU_RELEASE` to indicate that the menu item should be freed when the parent menu is destroyed.

11.9 Menu-generating Procedures

In certain situations, the menu items for a particular menu cannot be known ahead of time. For example, a mail application allows users to write mail messages to files in a designated *folder directory*. If a menu is going to display the current folders in that directory, then the menu items should be updated any time a folder is created or deleted from that directory. But rather than updating the folder at the time the directory contents change, it would be better to scan the directory and use each filename in the directory as a menu item.

For such situations, it is necessary to defer the creation of the folder menu until it needs to be displayed. Therefore, you still create the pullright menu item so the user can select the item. But rather than specifying a pullright menu associated with the item, specify a routine that will generate the menu. When the menu needs to be displayed, the routine is called which returns a menu.

To do this, you specify the attribute `MENU_GEN_PULLRIGHT` when creating the menu item, as shown below:

```
Menu menu, gen_folder_menu();
void change_to_folder();
```

```
menu = (Menu)xv_create(NULL, MENU,
    MENU_TITLE_ITEM,        "Mail Folders",
    MENU_NOTIFY_PROC,       change_to_folder,
    MENU_STRINGS,           "/usr/spool/mail", "~/mbox", NULL,
    MENU_ITEM,
        MENU_STRING,        "~/Mail"
        MENU_GEN_PULLRIGHT, gen_folder_menu,
        NULL,
    NULL);
```

There is a shortcut attribute that allows you to specify both the menu item's string and the MENU_GEN_PULLRIGHT procedure in the same call. The attribute is MENU_GEN_PULLRIGHT_ITEM. It is used as follows:

```
MENU_GEN_PULLRIGHT_ITEM, "~/Mail", gen_folder_menu,
```

The menu-generating routine may do whatever is necessary to build a new menu, but you should be careful that the routine does not take too much processing time, since the user is waiting with the MENU button pressed for the menu to be displayed. Also remember that a pointer grab is going on, so the routine should avoid any interaction with the user (such as error dialog boxes).

The form of the menu-generating procedure is:

```
Menu
menu_gen_proc(menu_item, op)
    Menu_item menu_item;
    Menu_generate op;
```

This routine may be called *each time* the menu is needed. If the menu only needs to be created once, you can return to the same menu each time you need the menu.

The op parameter is one of the following enumerated types:

```
typedef enum {
    MENU_DISPLAY,
    MENU_DISPLAY_DONE,
    MENU_NOTIFY,
    MENU_NOTIFY_DONE
} Menu_generate;
```

op indicates the condition in which your routine has been called. The MENU_DISPLAY value indicates that the menu is going to be displayed while MENU_DISPLAY_DONE indicates that the menu has been displayed and dismissed. If the user makes a selection in the menu, the routine is called with MENU_NOTIFY before the menu's callback routine is called, then again with MENU_NOTIFY_DONE after the routine is called. If the user makes no selection, the latter two cases are not called. If they are called (the user did make a selection), then the latter two cases are called *after* MENU_DISPLAY_DONE is called.

Because you create your menus, you would think that you should destroy them as well. However, because of the unpredictable sequence of actions taken by the user, there is no way to determine when to free the menu. Therefore, you should never destroy menus at all. If the menu-generating procedure is called multiple times for the same menu, just reconstruct the menu from the same menu handle that you have.

Furthermore, the menu-generating routine must always return the same menu that it passed to you. You cannot return other menus to display. If the new menu is going to contain a completely different set of menu items, you should destroy all the menu items before creating the new list. As it is the same with PANEL_LIST items, menu items are destroyed in *reverse* order.

The special case for this problem is: what if there is no menu to redisplay again? In this case, you are allowed to build a new menu and return a handle to it. The following code shows an example, testing to see if there already is a menu associated with a particular pullright menu item.

```
Menu
menu_gen_proc(menu_item, op)
Menu_item menu_item;
Menu_generate op;
{
    int i;
    Menu menu;

    ...
    switch (op) {
        ...
        case MENU_DISPLAY :
            if (menu = (Menu)xv_get(menu_item, MENU_PULLRIGHT)) {
                /* first destroy old menu items */
                for (i = (int)xv_get(menu, MENU_NITEMS); i > 0; i--) {
                    xv_set(menu, MENU_REMOVE, i, NULL);
                    xv_destroy(xv_get(menu, MENU_NTH_ITEM, i));
                }
            else
                menu = (Menu)xv_create(NULL, menu, NULL);
            /* now rebuild the menu items */
        ...
    }
    ...
}
```

In the above code fragment, we are removing the menu items sequentially in reverse order by using the MENU_REMOVE attribute. We start with the last item and move to item 1. The first item, remember, is the title item, if it exists. If you want to retain this item, stop at menu item 2.

The sample program *menu_dir2.c* in Appendix F, *Example Programs*, demonstrates how a menu-generating routine is used.

A debugging hint: If your menu-generating routine generates a run-time error, be careful when trying to debug the program under a debugger. The problem is that when you run the program in a debugger and the program generates a run-time error, the debugger will stop execution and wait for input. In the meantime, the server has a pointer grab so keyboard focus is directed to the menu's window which is not able to receive input.

At this point, there is no way to interact with any program on the console—you will have to go to another server, computer or terminal connected to your workstation and kill the debugger (this will terminate the program and release the pointer grab). You may think of more clever ways to handle this situation depending on your workstation configuration, but

the point is that you should be aware of the extremely inconvenient side effects whenever you play with server grabs.

11.9.1 Parent Menus

Recall that the menu notification routines take two parameters: the menu that was popped up and the menu item that was selected. However, if the user chose an item from a long cascade of pullright menus, it may be necessary to determine the initial (root) menu of the cascade. To support this, the attribute MENU_PARENT is used to get the owner of a menu or menu item.

This attribute can only be used with xv_get(). When MENU_PARENT is used with a menu item, xv_get() returns the handle of the enclosing menu.

```
Menu menu;

menu = xv_get(item, MENU_PARENT);
```

On the other hand, if xv_get() is passed a menu, the menu item returned is the menu item from which the submenu was pulled-right.

```
Menu_item item;

item = xv_get(menu, MENU_PARENT);
```

If the item returned is NULL, the menu is the root menu.

MENU_PARENT is only valid while the menu is active. Since menus can be shared, saying that a menu's parent is the one who uses it as a MENU_PULLRIGHT is not valid, since many menus could have that one menu as a MENU_PULLRIGHT.

The following code fragment shows how the entire menu cascade is traversed, starting from the leaf of the menu tree (the item the user selected).

```
Menu menu, item;

while (item = (Xv_opaque)xv_get(menu, MENU_PARENT))
    if ((Xv_pkg *)xv_get(item, XV_TYPE) != MENUITEM)
        break;
    else
        menu = xv_get(item, MENU_PARENT);
```

The above loop starts by getting the parent of an arbitrary menu. This menu could be the menu parameter in a menu item's callback routine. If the parent menu returned is NULL, then the menu is already the top level menu. Otherwise, get the type of the object returned. If the menu is a pullright menu, then the parent of the menu should be a MENUITEM (since its pullright is a menu). If it is not, then it could be a server object. Whatever it is, we have reached the top level of the menu cascade and should break out of the loop.

11.9.2 Using MENU_GEN_PROC

MENU_GEN_PROC specifies a function that is used to modify, add, or delete menu items from the menu whose handle is passed to the procedure. The *op* argument tells the state of the menu when the function is called. The argument *op* is one of the values: MENU_DISPLAY, MENU_DISPLAY_DONE, MENU_NOTIFY, or MENU_NOTIFY_DONE as defined by **Menu_generate** in *openmenu.h*. You do not destroy the menu itself. If you do not know what the item will show as text or as an image at the time the menu is created or if there is other unknown information, you can defer the creation of the menu item until the item is actually needed by specifying the item creation routine.

11.10 Using Toggle Menus

Toggle menus are menus with nonexclusive settings. The user can toggle menu items, turning them *on* or *off*. More than one menu item may be selected at a time. The only difference for creating these menu items is that MENU_TOGGLE_MENU is used as the *package* parameter to xv_create() and that menu items may not have pullright menus associated with them. Therefore, these are typically simple menus.

In the code below, we build a toggle menu that has three items in it. If the menu has been displayed and the user makes a selection, on or off, the notification routine is called no differently from any other menu notification procedure:

```
void toggle_bold(), toggle_size(), toggle_italic();
Menu menu;

menu = (Menu)xv_create(NULL, MENU_TOGGLE_MENU,
    MENU_TITLE_ITEM,    "Text Rendering",
    MENU_ACTION_ITEM,   "Bold Style",     toggle_bold,
    MENU_ACTION_ITEM,   "Large Font",     toggle_size,
    MENU_ACTION_ITEM,   "Italics",        toggle_italic,
    NULL);
```

In this case, we are specifying three different notify procedures for each menu item. Since each performs a completely separate function, the menu items need not call the same routine.

To determine exactly which menu items are selected, you must loop through all the items in the menu:

```
toggle_notify(menu, item)
Menu menu
Menu_item item;
{
    int i;

    for (i = (int)xv_get(font_menu, MENU_NITEMS); i > 0; i--)
        if (xv_get(xv_get(font_menu, MENU_NTH_ITEM, i),
                   MENU_SELECTED)) {
            printf("item %d selected\n", i);
            /* do whatever other processing may need to be done */
        }
}
```

This loop starts at the last item and works towards the first. The first item starts at 1, not 0; the *0th* item is the menu's title item and cannot be retrieved.

11.11 Menu Layout

By default, pop-up menus place their items vertically. If there are too many items, a new column may be started in order to display the entire menu on the screen. You can specify the number of rows and columns for the menu by using the attributes `MENU_NROWS` and `MENU_NCOLS`.

Although specifying menu item layout is certainly legal and acceptable to OPEN LOOK, explicit menu item layout should be avoided for anything other than static menus. Dynamic menus will have problems maintaining menu item order, and if you use a pin-up menu, the command frame will almost certainly not match the appearance of the menu. To guarantee that your pin-up menu looks the same as the menu, specify your own pin-up procedure. (See the following section for more information.)

11.12 Making Pin-up Menus

As the programmer, you may give the user the option of pinning up a menu by providing the pushpin in the pop-up menu. To accomplish this, XView provides the attribute `MENU_GEN_PIN_WINDOW`. If specified, XView generates the pin window frame automatically by creating a command frame, a panel and a series of panel items that correspond to the menu items. These pin window components are actually created the first time the user pulls down the menu and pushes the pin in. You cannot use `xv_get` to retrieve the command frame for the menu until after the menu is pinned. This new frame is dynamic, so any changes to the menu are reflected in the pinup frame provided it is not currently being displayed. If the pinup frame is currently being displayed and the menu contents change, the pinned menu will not reflect the new changes. The changes will appear the next time the menu is pinned.

Since menu items are translated into panel items in a pinned menu, programmers should not allow more than 32 unique values for a pinned menu (32 is the size of an **unsigned int** on most machines).

MENU_GEN_PIN_WINDOW takes two values as parameters. One is a base frame; the other is a string that acts as a title for the frame. The menu's pin window is sized according to the width of the widest menu item, not according to the title. You should choose a title that will fit within the size of the pin window. If the title specified in MENU_GEN_PIN_WINDOW is too long, then it will be truncated. Example 11-2 shows how to create a menu with a pushpin.

Example 11-2. How to create a menu containing a pushpin

```
Frame  frame;
Menu   menu;
void   func1(), func2();
...
/* Create base frame for the application */
frame = (Frame)xv_create(NULL, FRAME, NULL);
...
menu = (Menu)xv_create(NULL, MENU,
    /* the pinup menu subframe is a child of the base frame */
    MENU_GEN_PIN_WINDOW,    frame,  "title",
    MENU_ITEM,
        MENU_STRING,       "item1",
        MENU_NOTIFY_PROC,  func1,
        NULL,
    MENU_ITEM,
        MENU_STRING,       "item2",
        MENU_NOTIFY_PROC,  func2,
        NULL,
    NULL);
...
```


The new command frame is created as a subframe of the `frame` value. The title label for the menu and the frame is the *title* value. Note that you should not use MENU_TITLE_ITEM if you are using MENU_GEN_PIN_WINDOW since a menu item is automatically inserted at the top of the menu. If this item is removed, the pin will also be removed.

When the menu is displayed as a result of a call to `menu_show()`, a pushpin in the upper-left corner of the menu is displayed allowing the user to pin up the menu, that causes the menu to go away and the subframe to be displayed. You can get a handle to this subframe if you need it by calling:

```
Frame subframe = (Frame)xv_get(menu, MENU_PIN_WINDOW);
```

You can get a handle to the panel associated with that frame by calling:

```
Panel panel = (Panel)xv_get(frame, FRAME_CMD_PANEL);
```

If you choose to write your own pin window-generating procedures, there are several attributes that you might find helpful in implementing your routines:

MENU_PIN — This Boolean attribute indicates that the menu has a pushpin. If an application removes the first menu item, which is the title for the pinned menu, then the menu's pin will also be removed.

MENU_PIN_WINDOW — This attribute assigns the command frame you created to the menu's pin window. Once the pushpin is pushed in, the MENU package automatically sets the command frame's XV_SHOW attribute to TRUE, allowing the frame to be displayed.

MENU_PIN_PROC This attribute provides the menu with a procedure that is called when the user pushes the pin in. You may override the default procedure that shows the pin_window of the menu by providing your own routine using this attribute. This routine overrides the default behavior of setting XV_SHOW on the command frame, so this responsibility lies in this routine.

MENU_DONE_PROC This routine is called whenever a pop-up menu has been taken down after a menu item has been selected, pinned up or simply dismissed without a selection being made. Again, this overrides the default action of setting XV_SHOW to FALSE, so this responsibility lies with the MENU_DONE_PROC routine.

11.13 Notification Procedures

When a menu item is selected, a notification procedure is called to notify the host application that the user has made a selection. If the menu item does not have a notify procedure, the parent menu's notification procedure is called instead. If the menu does not have a notification procedure, no action is taken. Be sure that each menu item or its parent menu has a MENU_NOTIFY_PROC routine associated with it. Otherwise, choosing a menu item has no effect. If you wish to make a menu item *inactive*, you should set the attribute MENU_INACTIVE to TRUE.

The primary difference between the MENU package and the PANEL package with respect to the notification mechanism is that if a menu item has no notification procedure associated with it, the notify procedure of the parent menu is used. The PANEL package has no such feature. To differentiate between the notify procedure of a *menu* and the notify procedure of a *menu item*, the term *action procedure* is sometimes used to refer to the menu item's notify procedure. Thus, you may come across the term *action procedure* or the attribute MENU_ACTION_PROC outside of this manual.

However, because there is functionally no difference between the MENU_ACTION_PROC and the MENU_NOTIFY_PROC and since all other XView objects use notify procedures to register their callbacks, we are going to attempt to maintain consistency and avoid potential confusion with the terms by referring to both notify procedures commonly using MENU_NOTIFY_PROC.

The form of the callback routine is:

```
void
menu_notify_proc(menu, menu_item)
    Menu menu;
    Menu_item menu_item;
```

The program in Example 11-3 demonstrates several of the concepts introduced in the chapter so far. *xv_menu.c* creates a simple frame, a canvas, and a pop-up menu. The menu is a *static* menu because all the menu items are created in-line with the menu-generating procedure. The menu itself has a notify procedure which is called if any of the menu items specified by MENU_STRINGS are selected. The result is to display the text of the selected item in the

header of the frame. An additional menu item is specified that has a pullright menu that can be pinned up.

Example 11-3. The xv_menu.c program

```
/*
 * xv_menu.c -
 *      Demonstrate the use of an XView menu in a canvas subwindow.
 *      Menu is brought up with right mouse button and the selected
 *      choice is displayed in the canvas.  Allows menu to be pinned.
 */
#include <xview/xview.h>
#include <xview/canvas.h>

Frame   frame;

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

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

    frame = (Frame)xv_create(NULL, FRAME,
        FRAME_LABEL,    argv[0],
        NULL);
    canvas = (Canvas)xv_create(frame, CANVAS,
        XV_WIDTH,       300,
        XV_HEIGHT,      200,
        NULL);
    menu = (Menu)xv_create(NULL, MENU,
        MENU_TITLE_ITEM,        "Junk",
        MENU_STRINGS,           "Yes", "No", "Maybe", NULL,
        MENU_NOTIFY_PROC,       my_notify_proc,
        MENU_ITEM,
            MENU_STRING,        "Save",
            MENU_NOTIFY_PROC,   my_notify_proc,
            MENU_PULLRIGHT,
                xv_create(canvas, MENU,
                    MENU_GEN_PIN_WINDOW,        frame, "Save",
                    MENU_ITEM,
                        MENU_STRING,            "Update Changes",
                        MENU_NOTIFY_PROC,       my_notify_proc,
                        NULL,
                    NULL),
            NULL,
        MENU_ITEM,
            MENU_STRING,        "Quit",
            MENU_NOTIFY_PROC,   exit,
            NULL,
        NULL);

    xv_set(canvas_paint_window(canvas),
```

Example 11-3. The xv_menu.c program (continued)

```
        WIN_CONSUME_EVENTS,     WIN_MOUSE_BUTTONS, NULL,
        WIN_EVENT_PROC,         my_event_proc,
        /* associate the menu to the canvas win for easy retrieval */
        WIN_CLIENT_DATA,        menu,
        NULL);

    window_fit(frame);
    window_main_loop(frame);
}

/*
 * my_notify_proc - Display menu selection in frame header.
 */
void
my_notify_proc(menu, menu_item)
Menu menu;
Menu_item menu_item;
{
    xv_set(frame,
        FRAME_LABEL,    xv_get(menu_item, MENU_STRING),
        NULL);
}

/*
 * Call menu_show() to display menu on right mouse button push.
 */
void
my_event_proc(window, event)
Xv_Window window;
Event *event;
{
    if (event_action(event) == ACTION_MENU && event_is_down(event)) {
        Menu menu = (Menu)xv_get(window, WIN_CLIENT_DATA);
        menu_show(menu, window, event, NULL);
    }
}
```

The output of Example 11-3 is shown in Figure 11-8.

11.14 Finding Menu Items

You can use `xv_find()` to locate menu items that match certain attribute-value pairs. The form of using `xv_find()` for menus is:

```
Menu_item item;

item = (Menu_item)xv_find(menu, MENUITEM,
    <attribute-value list>,
    NULL);
```

`xv_find()` returns menu items, not menus. By default, when you search for items, each item in a menu is searched before descending into a menu item's pullright menu, should it

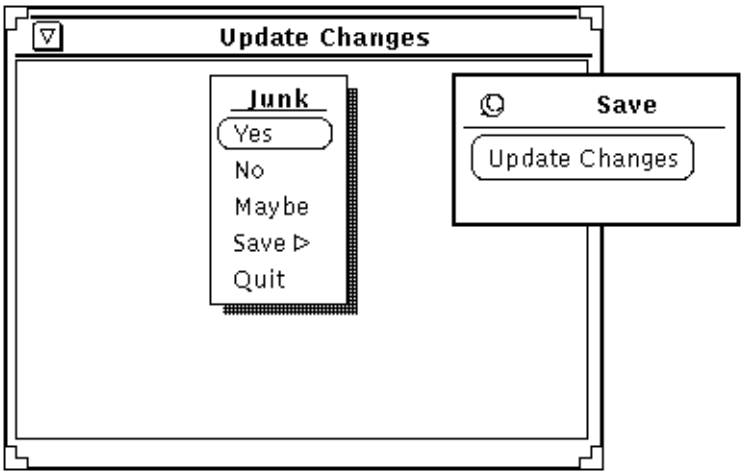

Figure 11-8. Output of xv_menu.c

exist. However, you can override this default behavior by specifying the attribute `MENU_DESCEND_FIRST`. During a menu search, if an item with a pullright is found, then this attribute indicates whether the search should continue through the pullright or to the next item in the current menu.

If a menu item or a menu item's pullright is a *generate* procedure, the generate procedure is called despite the fact that the menu or menu item will not be displayed. No matter how many attributes are given, `xv_find()` will return the first item found that matches all given attributes even though the item may have more attributes associated with it.

When specifying attribute-value pairs, you specify attributes in the same way as when you use `xv_create()`. For example, if you want to find a menu item with the string value of "fonts" and the callback routine of `my_notify_proc`, you would use:

```
menu_item = (Menu_item)xv_find(menu, MENUITEM,
    MENU_STRING,       "fonts",
    MENU_NOTIFY_PROC,  my_notify_proc,
    NULL);
```

Unless the attribute `XV_AUTO_CREATE` is set to `FALSE`, if `xv_find()` does not find the menu item that you are looking for, a new menu item will be created.

11.15 Initial and Default Menu Selections

Two special menu items are the *default item* (MENU_DEFAULT_ITEM) and the *selected item* (MENU_SELECTED_ITEM). The default item defaults to the *first* item in the menu, and the *selected item* is the selected item (or items for MENU_TOGGLE_MENU menus).

Although the default menu item may be set by using xv_set(), the user may interactively change the default menu item by holding down the CONTROL key while also selecting a menu item with the MENU button. Therefore, if the user selects a menu item that has a pullright menu, *but the pullright menu is not activated*,* when your notify procedure is called, you may choose to descend into the pullright menu and find the default menu item and call that item's callback routine.

11.16 Unpinned Command Frame Dismissal

XView normally handles unpinned command frame dismissal for you when a user action within the command frame completes successfully. If a menu is brought up from a Menu Button in the command frame, and the user makes a selection from the menu, the command frame is dismissed if the pushpin (if visible) is out. By default, the attribute MENU_NOTIFY_STATUS is set to XV_OK, which indicates that the command frame should be dismissed if the pushpin is out, and the callback returns successfully. However, if the user-specified action does not complete successfully, you may not want the command frame to be dismissed. In this case, within the menu's notify procedure or within the menu item's notify procedure, you should set the value of MENU_NOTIFY_STATUS to XV_ERROR. This indicates that the user selection was invalid or failed, and prevents the command frame from being dismissed.

11.17 Destroying Menus

Destruction of menus is an important task because menus are frequently used and, if their resources are not freed adequately, you could find the size of your application growing rapidly until your system runs out of available memory. Therefore, proper cleanup of menu destruction is imperative. Menus are destroyed using xv_destroy(). In the case of *static menus*, nothing more is required than calling xv_destroy(). This is because the internals of XView automatically set attributes discussed in this section.

Be aware of several situations, such as when you:

- Allocate your own strings or server images as menu item labels.
- Create your own menu items using xv_create(NULL, MENUITEM, . . .).
- Generate your own pullright menus.

*This might happen if the user did not drag the mouse far enough to the right.

The destruction phase walks down each menu item in the menu and tests each menu item to see if it has the MENU_RELEASE attribute set. This is not a Boolean attribute—it has no value associated with it at all. If you specify the attribute, the attribute is set. If you do not specify it, then the attribute is not set. As noted, menu items that have been created in-line have MENU_RELEASE set already.

Menu items that you create yourself do not have MENU_RELEASE set by default. You also may or may not *want* it set. If you plan to reuse menu items—a need that is common—then you do not want to set this attribute. However, you must maintain a handle to the menu item or it is lost. If the attribute is set, then the menu item is freed, but no other data associated with the menu item is destroyed. Only the item itself is. If you have any allocated data associated with the menu item, then you either need to free it yourself or give a hint to XView to free it for you.

The following subsections discuss other data allocated for menu items. Remember that freeing menus and menu items is not done automatically; this only happens as a result of your calling xv_destroy(). So, if you decide to free menus or menu items, you should be sure to free pullright menus and/or client data yourself beforehand.

There are cases when xv_destroy() will not remove a menu. In order to free the memory associated with a menu using xv_destroy(), you need to be certain that no objects reference the menu. For example, if you attach a menu to a panel button item using the attribute PANEL_ITEM_MENU, you need to be sure to clear the PANEL_ITEM_MENU attribute *before* you try to destroy the menu. In this example, the following calls would be required to clear the panel button item's attached menu, and to destroy the menu. For more information on this topic, refer to the description of XV_REF_COUNT in Chapter 7, *Panels*.

```
xv_set(panel_item, PANEL_ITEM_MENU, NULL, NULL);
xv_destroy(menu);
```


11.17.1 Freeing Allocated Strings

If you create a menu item with allocated data, you should *not* use them in a MENU_STRINGS list. Instead, you should create the menu items individually, as shown in Example 11-4.

Example 11-4. Creating individual menu items

```
char *str1;

if (str1 = malloc(strlen(buf)+1))
    strcpy(str1, buf);

menu = xv_create(NULL, MENU,
    MENU_ITEM,
        MENU_STRING,   str1,
        MENU_RELEASE_IMAGE,
        NULL,
    NULL);
```

The code in Example 11-4 shows a menu item that is created in-line because it is created using the MENU_ITEM attribute. However, because the string used as the menu item's label is allocated, we need to provide XView with a hint to release this data.

Similarly, if we used `xv_create()` to create a `Server_image` as the menu item's label, the `MENU_RELEASE_IMAGE` attribute suffices to free that data as well.

11.17.2 Freeing Pullright Menus

Even though a menu item has `MENU_RELEASE` set, if a pullright menu is associated with it, the menu will not be freed. In many cases, this is acceptable because many menu items may share the same pullright menu. If you are sure you do not need the menu anymore, then you should free it. Note that freeing the menu will attempt to free the menu items within it.

This is most commonly done in menu-generating routines installed as the `MENU_GEN_PULLRIGHT` attribute.

11.17.3 Menu Client Data

If a menu item is freed, you should be sure to free any client data that is associated with it. Client data may have been attached to the menu item using `XV_KEY_DATA` or `MENU_CLIENT_DATA`.

If you created menus for panel buttons, and you destroy the MENU button (or the panel associated with that button), then you are responsible for destroying the menu you created. The panel does not handle this for you. Destroying the menu attached to menu buttons is done the same way as it is for menus.

11.18 Example Program

The following brief descriptions are introductory notes about the programs *menu_dir.c* (listed in Example 11-5) and *menu_dir2.c* (listed in Appendix F, *Example Programs*). The comments in the programs as well as the code itself should be read for full details.

menu_dir.c demonstrates many of the features of the `MENU` package presented in this chapter. It displays a menu that contains all the files from the current directory. If a pathname is given on the command line, that directory is used. The entire menu hierarchy is built initially at start-up time, so directories that do not have extremely long paths should be specified.*

For each directory found, a new menu is created and the directory is descended building items for the new menu. *menu_dir2.c* also builds cascading menus for directories, but instead of descending into the directory tree, a menu-generating routine is called only if the user tries to go into a pullright.

Example 11-5. The menu_dir.c program

```
/*
```

*Don't even think of specifying /.

Example 11-5. The menu_dir.c program (continued)

```
 * menu_dir.c -
 * Demonstrate the use of an XView menu in a canvas subwindow.
 * A menu is brought up with the MENU mouse button and displays
 * menu choices representing the files in the directory.  If a
 * directory entry is found, a new pullright item is created with
 * that subdir as the pullright menu's contents.  This implementation
 * creates the entire directory tree initially.  Do not attempt to
 * build a tree from /.  You will most likely run out of resources.
 *
 * argv[1] indicates which directory to start from.
 */
#include <xview/xview.h>
#include <xview/canvas.h>
#include <sys/stat.h>
#include <sys/dir.h>
#include <X11/Xos.h>
#ifndef MAXPATHLEN
#include <sys/param.h> /* probably sun/BSD specific */
#endif /* MAXPATHLEN */

Frame   frame;

/*
 * main -
 *      Create a frame, canvas and menu.
 *      A canvas receives input in its canvas_paint_window().
 *      Its callback procedure calls menu_show().
 */
main(argc,argv)
int     argc;
char    *argv[ ];
{
    Canvas      canvas;
    extern void exit();
    void        my_event_proc();
    Menu        menu;
    Menu_item   mi, add_path_to_menu();

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

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

    mi = add_path_to_menu(argc > 1? argv[1] : ".");
    menu = (Menu)xv_get(mi, MENU_PULLRIGHT);

    /* associate the menu to the canvas win for easy retrieval */
    xv_set(canvas_paint_window(canvas),
        WIN_CONSUME_EVENTS,     WIN_MOUSE_BUTTONS, NULL,
```

Example 11-5. The menu_dir.c program (continued)

```
        WIN_EVENT_PROC,         my_event_proc,
        WIN_CLIENT_DATA,        menu,
        NULL);

    window_fit(frame);
    window_main_loop(frame);
}

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

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

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

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

/*
 * The path passed in is scanned via readdir().  For each file in the
 * path, a menu item is created and inserted into a new menu.  That
 * new menu is made the PULLRIGHT_MENU of a newly created panel item
 * for the path item originally passed it.  Since this routine is
```

Example 11-5. The menu_dir.c program (continued)

```
 * recursive, a new menu is created for each subdirectory under the
 * original path.
 */
Menu_item
add_path_to_menu(path)
char *path;
{
    DIR                 *dirp;
    struct direct       *dp;
    struct stat         s_buf;
    Menu_item           mi;
    Menu                next_menu;
    char                buf[MAXPATHLEN];

    /* don't add a folder to the list if user can't read it */
    if (stat(path, &s_buf) == -1 || !(s_buf.st_mode & S_IREAD))
        return NULL;
    if (s_buf.st_mode & S_IFDIR) {
        int cnt = 0;
        if (!(dirp = opendir(path)))
            /* don't bother adding to list if we can't scan it */
            return NULL;
        next_menu = (Menu)xv_create(XV_NULL, MENU, NULL);
        while (dp = readdir(dirp))
            if (strcmp(dp->d_name, ".") && strcmp(dp->d_name, "..")) {
                (void) sprintf(buf, "%s/%s", path, dp->d_name);
                if (!(mi = add_path_to_menu(buf)))
                    /* unreadable file or dir - deactivate item */
                    mi = xv_create(XV_NULL, MENUITEM,
                        MENU_STRING,         getfilename(dp->d_name),
                        MENU_RELEASE,
                        MENU_RELEASE_IMAGE,
                        MENU_INACTIVE,       TRUE,
                        NULL);
                xv_set(next_menu, MENU_APPEND_ITEM, mi, NULL);
                cnt++;
            }
        closedir(dirp);
        mi = xv_create(XV_NULL, MENUITEM,
            MENU_STRING,        getfilename(path),
            MENU_RELEASE,
            MENU_RELEASE_IMAGE,
            MENU_NOTIFY_PROC,   my_action_proc,
            NULL);
        if (!cnt) {
            xv_destroy(next_menu);
            /* An empty or unsearchable directory - deactivate item */
            xv_set(mi, MENU_INACTIVE, TRUE, NULL);
        } else {
            xv_set(next_menu, MENU_TITLE_ITEM, getfilename(path), NULL);
            xv_set(mi, MENU_PULLRIGHT, next_menu, NULL);
        }
        return mi;
    }
    return (Menu_item)xv_create(NULL, MENUITEM,
        MENU_STRING,            getfilename(path),
```

Example 11-5. The menu_dir.c program (continued)

```
        MENU_RELEASE,
        MENU_RELEASE_IMAGE,
        MENU_NOTIFY_PROC,       my_action_proc,
        NULL);
}
```

11.19 Menu Package Summary

Table 11-1 lists the procedures and macros in the `MENU` package. Table 11-2 lists the attributes in the `MENU` package. This information is described fully in the *XView Reference Manual*.

Table 11-1. Menu Procedures and Macros

`MENUITEM_SPACE()`
`menu_return_item()`
`menu_return_value()`
`menu_show`

Table 11-2. Menu Attributes

`MENU_APPEND_ITEM`	`MENU_NOTIFY_STATUS`
`MENU_CLASS`	`MENU_NROWS`
`MENU_COLOR`	`MENU_NTH_ITEM`
`MENU_CLIENT_DATA`	`MENU_PARENT`
`MENU_COL_MAJOR`	`MENU_PIN`
`MENU_DEFAULT`	`MENU_PIN_PROC`
`MENU_DEFAULT_ITEM`	`MENU_PIN_WINDOW`
`MENU_DESCEND_FIRST`	`MENU_PULLRIGHT`
`MENU_DONE_PROC`	`MENU_RELEASE`
`MENU_FIRST_EVENT`	`MENU_RELEASE_IMAGE`
`MENU_GEN_PIN_WINDOW`	`MENU_REMOVE`
`MENU_GEN_PROC`	`MENU_REMOVE_ITEM`
`MENU_GEN_PULLRIGHT`	`MENU_REPLACE`
`MENU_IMAGE`	`MENU_REPLACE_ITEM`
`MENU_IMAGES`	`MENU_SELECTED`
`MENU_INACTIVE`	`MENU_SELECTED_ITEM`
`MENU_INSERT`	`MENU_STRING`
`MENU_INSERT_ITEM`	`MENU_STRINGS`
`MENU_ITEM`	`MENU_TITLE`
`MENU_LAST_EVENT`	`MENU_TITLE_ITEM`

Table 11-2. Menu Attributes (continued)

MENU_NCOLS	MENU_TYPE
MENU_NITEMS	MENU_VALID_RESULT
MENU_NOTIFY_PROC	MENU_VALUE
XV_DEPTH	XV_VISUAL
XV_VISUAL_CLASS	