B
The notice_prompt Function

This section describes the XView compatibility procedure `notice_prompt()`. If you need to create a new notice, use the `NOTICE` package described in Chapter 12, *Notices*. The information in this chapter describes the old notice interface that is supported for compatibility with older XView versions.

A notice is a pop-up window that notifies the user of a problem or asks a question that requires an immediate response. The notice grabs the entire screen so no other windows or applications can receive input until the user responds to the notice.

Notices are implemented using the `FULLSCREEN` package to grab the keyboard and pointer events from the server. (The `FULLSCREEN` package is described in Chapter 15, *Nonvisual Objects*.) The notice window, which owns the fullscreen object, is a nonrectangular transient X window with the X-window attribute `override_redirect`* set. When the notice is created, the notice window is immediately displayed. When the user responds to one of the available choices, the notice session ends.

The notice_prompt
Function

B.1 Creating and Displaying Notices

To use the `NOTICE` package in applications, the header file *<xview/notice.h>* must be included. Notices are special XView objects because they are not created via `xv_create()`. Also, they cannot be modified using `xv_set()`. Notices are created using the special procedure `notice_prompt()`:

```
int
notice_prompt(owner, event, attrs)
    Xv_Window   owner;
    Event      *event;
    attributes ...
```

When creating a notice, the owner must be a valid XView object that has a window associated with it. This can be a panel or a frame, but it is typically the window that causes the notice to be created. If the user tries to type in a read-only text subwindow, a notice might appear from that window informing the user of the error. The `event` might be `NULL` if you are not using `NOTICE_TRIGGER` (see Section 12.1.2, “Notice Triggers”).

*`override_redirect` tells the window manager to not provide window decorations.

Because the notice window is not a part of any other XView package and it does not allow window-specific attributes, you cannot use any generic, common or window attributes to configure the notice window; you can only use `NOTICE_*` attributes.

Notice windows are explicitly specified by OPEN LOOK and cannot be modified. If you wish to create a notice-type interface that is not OPEN LOOK compliant (which is not recommended), you need to learn more about the fullscreen object described in Chapter 15, *Non-visual Objects*.

Your application has control over the messages that are displayed in the notice window as well as the choices available to the user as responses. `notice_prompt()` creates a window, grabs the server, waits for the user to make a selection on one of the available button choices, then destroys the window. You never have a handle to the notice *object* itself—only the resulting choice made by the user. The result is the return value of the `notice_prompt()` function.

A very simple case of a notice prompt is demonstrated in Example B-1.

Example B-1. The simple_notice.c program

```
/*
 * simple_notice.c -- Demonstrate the use of notices.
 */
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/notice.h>

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

    /*
     * Initialize XView, create a frame, a panel and one panel button.
     */
    xv_init(XV_INIT_ARGS, argc, argv, NULL);

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

    /* make sure everything looks good */
    window_fit(panel);
    window_fit(frame);

    /* start window event processing */
    xv_main_loop(frame);
}

/*
```

Example B-1. The simple_notice.c program (continued)

```
 * my_notify_proc() -- called when the user selects the Quit button.
 *      The notice appears as a result of notice_prompt().  Here
 *      the user must chooses YES or NO to confirm or deny quitting.
 */
Xv_opaque
my_notify_proc(item, event)
Panel_item  item;
Event      *event;
{
    int         result;

    result = notice_prompt(panel, NULL,
        NOTICE_FOCUS_XY,        event_x(event), event_y(event),
        NOTICE_MESSAGE_STRINGS, "Do you really want to quit?", NULL,
        NOTICE_BUTTON_YES,      "Yes",
        NOTICE_BUTTON_NO,       "No",
        NULL);

    if (result == NOTICE_YES)
        exit(0);
}
```

The program *simple_notice.c* contains a panel with a Quit button. When the user selects the Quit button, a notice pops up to prompt the user for confirmation. What the user sees is shown in Figure B-1. If the user presses “Yes,” the program exits.

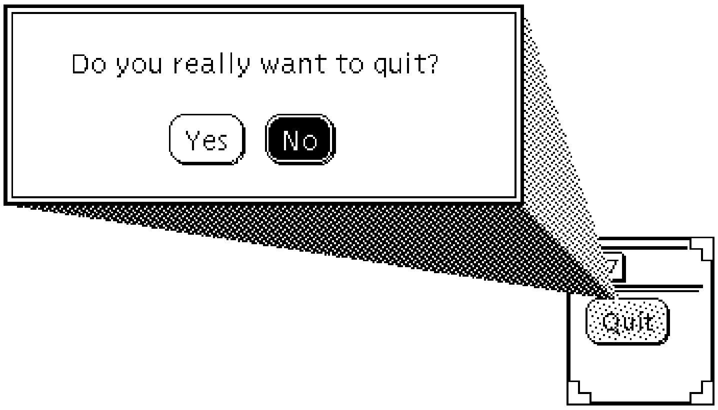

Figure B-1. Output of simple_notice.c while the notice is up

The position from which the notice shadow emanates is described by the attribute `NOTICE_FOCUS_XY`. This value defaults to the current mouse position when the application calls `notice_prompt()`. As shown in *simple_notice.c*, the point from which the notice shadow emanates appears to be the same position as the location where the panel button was selected. Due to possible delays with the X server, by the time the `notice_prompt()` routine gets called, the location of the mouse may have moved from the place where the panel button was selected. To be sure that the notice prompt appears to emanate from the original mouse-down location, we use the coordinates of the mouse position from the `event` structure. The values for `NOTICE_FOCUS_XY` are relative to the origin of the window passed as the first parameter to `notice_prompt()`.

B.1.1 Response Choices and Values

Two responses are normally present whenever a notice appears: "Yes" and "No." These are defined for convenience in <*xview/notice.h*>:

```
#define NOTICE_YES  1
#define NOTICE_NO   0
```

These are the return values that `notice_prompt()` might return that correspond directly to the attributes `NOTICE_BUTTON_YES` and `NOTICE_BUTTON_NO`. As shown in *simple_notice.c*, these are the only two choices made available to the user.

These two choices are special in another way: they respond to *accelerator* keys. That is, the RETURN key can be used instead of selecting the `NOTICE_BUTTON_YES` button with the pointer, and the STOP key can be used instead of selecting `NOTICE_BUTTON_NO`.

Also, when these choices are used, the cursor is immediately bound to the button associated with `NOTICE_BUTTON_YES` because this is the *default* response to the notice. As a hint to the programmer, it is always desirable to word all questions so the default answer is "Yes."

It is quite common for the application to have more than one appropriate response to some kind of notice prompt. Suppose that your application is an editor of some kind. If the user selects the Quit button and there have been changes to the file that have not been accounted for, you might wish to inform the user and allow more than one response: quit, updating changes; quit, ignoring changes; or cancel the quit all together. To implement these new choices, use the `NOTICE_BUTTON` attribute to define the choices available:

```
result = notice_prompt(panel, NULL,
    NOTICE_MESSAGE_STRINGS,
        "There have been modifications since your last update",
        "Would you like to quit or continue editing?",
        NULL,
    NOTICE_BUTTON,    "Quit, Update changes",    101,
    NOTICE_BUTTON,    "Quit, Ignore changes",    102,
    NOTICE_BUTTON,    "Continue Editing",        103,
    NULL);
```

The `NOTICE_BUTTON` attribute takes two parameters: the button label* and the return value if that button is selected. In this case, the possible return values for the call to `notice_prompt` are 101, 102 and 103 (in addition to possible error return values). The application should make its decision on how to proceed based on the return value.

*The button can display text only; no graphic images can be displayed.

Because the NOTICE_BUTTON attribute is used, there is no default choice and no accelerators associated with the notice; the user must use the pointer to select one of the available choices.

B.1.2 Notice Triggers

If you want to assign accelerators to notice buttons, or if you find it necessary to give the user the choice of using mouse buttons or keyboard events to respond to a notice, you can identify *triggers* that cause the notice to return. The value returned in this case is NOTICE_TRIGGERED, and the event that caused the trigger will be in the `Event *` passed in the call to `notice_prompt()`. When triggers are not used, the `Event *` can be NULL.

Example B-2 shows how one might use the NOTICE_TRIGGER to get a particular event:

Example B-2. The trigger_notice.c program

```
/*
 * trigger_notice.c -- Demonstrate the use of triggers in notices.
 */
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/notice.h>

main(argc,argv)
int     argc;
char    *argv[ ];
{
    Frame       frame;
    Panel       panel;
    Xv_opaque   my_notify_proc();
    extern void exit();

    /*
     * Initialize XView, create a frame, a panel and one panel button.
     */
    xv_init(XV_INIT_ARGS, 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,      exit,
        NULL);
    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Move",
        PANEL_NOTIFY_PROC,      my_notify_proc,
        NULL);

    /* make sure everything looks good */
    window_fit(panel);
    window_fit(frame);

    /* start window event processing */
    xv_main_loop(frame);
}
```

The notice_prompt Function

Example B-2. The trigger_notice.c program (continued)

```
/*
 * my_notify_proc() -- called when the user selects the "Move"
 * panel button.  Put up a notice_prompt to get new coordinates
 * to move the main window.
 */
Xv_opaque
my_notify_proc(item, event)
Panel_item  item;
Event      *event;
{
    int         result, x, y;
    Panel       panel = (Panel)xv_get(item, PANEL_PARENT_PANEL);
    Frame       frame = (Frame)xv_get(panel, XV_OWNER);

    x = event_x(event), y = event_y(event);
    printf("original click relative to panel: %d, %d0, x, y);
    result = notice_prompt(panel, event,
        NOTICE_FOCUS_XY,        x, y,
        NOTICE_MESSAGE_STRINGS,
            "You may move the window to a new location specified by",
            "clicking the Left Mouse Button somewhere on the screen",
            "or cancel this operation by selecting
            NULL,
        NOTICE_BUTTON_YES,      "cancel",
        NOTICE_TRIGGER,         MS_LEFT,
        NOTICE_NO_BEEPING,      TRUE,
        NULL);

    if (result == NOTICE_TRIGGERED) {
        x = event_x(event) + (int)xv_get(frame, XV_X);
        y = event_y(event) + (int)xv_get(frame, XV_Y);
        printf("screen x,y: %d, %d0, x, y);
        xv_set(frame, XV_X, x, XV_Y, y, NULL);
    }
}
```

When this program is run and the user selects the Move panel button, a notice is displayed instructing the user to select a new position for the application window. When the user selects a new location, the window frame moves to that position. Note that the window manager adds a title bar and other decorations around the frame; do not expect the upper-left corner of the frame to move to the new position. The real frame's origin is moved to the new position, and the frame's decorations are moved as well but not aligned to the same values (it will be somewhat higher).

When `notice_prompt()` returns, the `Event` structure that was passed to it contains the event that triggered the notice to return. The *x* and *y* coordinates in the `Event` structure are relative to the origin of the notice owner window.

To translate these coordinates to screen-specific coordinates, save the original event location and add to that the (*x*, *y*) coordinates returned when `notice_prompt()` returns as well as the current coordinates of the frame (main application).

Before leaving *trigger_notice.c*, we should mention the attribute NOTICE_NO_BEEPING that is used to prevent the notice from beeping when it is displayed. Beeping the screen is usually done when there is an error condition you wish to alert the user about. In this example, there is no error condition—it is a simple dialog with the user.

B.2 Another Example

In the previous example, we used many of the attributes covered in this section in addition to using some generic and common attributes for the panel items. Example B-3 goes a little further to demonstrate how the NOTICE package works in conjunction with the rest of XView. It creates a frame, a panel with two panel buttons and a message item. Initially, only the Quit button and the Commit button are displayed. When the user selects either button, a notice pops up asking the user to confirm or cancel the proposed action. If the user confirms quitting the program, the program quits. Otherwise, the result, either Confirmed or Canceled, is displayed as the text of the message item.

Example B-3. The notice.c program

```
/*
 * notice.c --
 * This application creates a frame, a panel, and 3 panel buttons.
 * A message button, a Quit button (to exit the program) and a
 * dummy "commit" button.  Extra data is attached to the panel
 * items by the use of XV_KEY_DATA.  The callback routine for the
 * Quit and Commit buttons is generalized enough that it can apply
 * to either button (or any arbitrary button) because it extracts
 * the expected "data" (via XV_KEY_DATA) from whatever panel
 * button might have called it.
 */
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/notice.h>

/*
 * assign "data" to panel items using XV_KEY_DATA ... attach the
 * message panel item, a prompt string specific for the panel
 * item's notice_prompt, and a callback function if the user
 * chooses "yes".
 */
#define MSG_ITEM        10 /* any arbitrary integer */
#define NOTICE_PROMPT   11
#define CALLBACK_FUNC   12

main(argc,argv)
int     argc;
char    *argv[ ];
{
    Frame       frame;
    Panel       panel;
    Panel_item  msg_item;
    Xv_opaque   my_notify_proc();
    extern int  exit();
```

The notice_prompt Function

Example B-3. The notice.c program (continued)

```
    /*
     * Initialize XView, and create frame, panel and buttons.
     */
    xv_init(XV_INIT_ARGS, argc, argv, NULL);
    frame = (Frame)xv_create(XV_NULL, FRAME,
        FRAME_LABEL,            argv[0],
        NULL);
    panel = (Panel)xv_create(frame, PANEL,
        PANEL_LAYOUT,           PANEL_VERTICAL,
        NULL);
    msg_item = (Panel_item)xv_create(panel, PANEL_MESSAGE, NULL);
    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Quit",
        PANEL_NOTIFY_PROC,      my_notify_proc,
        XV_KEY_DATA,            MSG_ITEM,       msg_item,
        /*
         * attach a prompt specific for this button used by
         * notice_prompt()
         */
        XV_KEY_DATA,            NOTICE_PROMPT,  "Really Quit?",
        /*
         * a callback function to call if the user answers "yes"
         * to prompt
         */
        XV_KEY_DATA,            CALLBACK_FUNC,  exit,
        NULL);
    /*
     * now that the Quit button is under the message item,
     * layout horizontally
     */
    xv_set(panel, PANEL_LAYOUT, PANEL_HORIZONTAL, NULL);
    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Commit...",
        PANEL_NOTIFY_PROC,      my_notify_proc,
        XV_KEY_DATA,            MSG_ITEM,       msg_item,
        /*
         * attach a prompt specific for this button used by
         * notice_prompt()
         */
        XV_KEY_DATA,            NOTICE_PROMPT,  "Update all changes?",
        /*
         * Note there is no callback func here, but one could be
         * written
         */
        NULL);

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

/*
 * my_notify_proc()
 * The notice appears as a result of notice_prompt().
 * The "key data" associated with the panel item is extracted via
```

Example B-3. The notice.c program (continued)

```
 * xv_get().  The resulting choice is displayed in the panel
 * message item.
 */
Xv_opaque
my_notify_proc(item, event)
Panel_item  item;
Event       *event;
{
    int          result;
    int        (*func)();
    char        *prompt;
    Panel_item   msg_item;
    Panel        panel;

    func = (int(*)())xv_get(item, XV_KEY_DATA, CALLBACK_FUNC);
    prompt = (char *)xv_get(item, XV_KEY_DATA, NOTICE_PROMPT);
    msg_item = (Panel_item)xv_get(item, XV_KEY_DATA, MSG_ITEM);
    panel = (Panel)xv_get(item, PANEL_PARENT_PANEL);
    /*
     *  Create the notice and get a response.
     */
    result = notice_prompt(panel, NULL,
        NOTICE_FOCUS_XY,        event_x(event), event_y(event),
        NOTICE_MESSAGE_STRINGS,
                prompt,
                "Press YES to confirm",
                "Press NO to cancel",
                NULL,
        NOTICE_BUTTON_YES,      "YES",
        NOTICE_BUTTON_NO,       "NO",
        NULL);

    switch(result) {
        case NOTICE_YES:
            xv_set(msg_item, PANEL_LABEL_STRING, "Confirmed", NULL);
            if (func)
                (*func)();
            break;
        case NOTICE_NO:
            xv_set(msg_item, PANEL_LABEL_STRING, "Cancelled", NULL);
            break;
        case NOTICE_FAILED:
            xv_set(msg_item, PANEL_LABEL_STRING, "unable to pop-up",
              NULL);
            break;
        default:
            xv_set(msg_item, PANEL_LABEL_STRING, "unknown choice",
              NULL);
    }
}
```

B.3 Notice Package Summary

Table B-1 lists the attributes, procedures and macros for the `NOTICE` package. This information is described fully in the appendices of the *XView Reference Manual*.

Table B-1. Notice Attributes, Procedures, and Macros

Attributes	Procedures and Macros
NOTICE_BUTTON NOTICE_BUTTON_NO NOTICE_BUTTON_YES NOTICE_FOCUS_XY NOTICE_FONT NOTICE_MESSAGE_STRINGS NOTICE_MESSAGE_STRINGS_ARRAY_PTR NOTICE_NO_BEEPING NOTICE_TRIGGER	notice_prompt()