12
Notices

A notice is a pop-up window that notifies the user of a problem or asks a question that requires a response. Generally, notices report serious warnings or errors. OPEN LOOK notices do not have headers or footers and cannot be moved. The XView notice object is subclassed from the XView generic object. As with any XView object, you can configure a notice using attributes and you can use `xv_create()`, `xv_get()`, and `xv_set()`. Figure 12-1 shows the notice object class hierarchy.

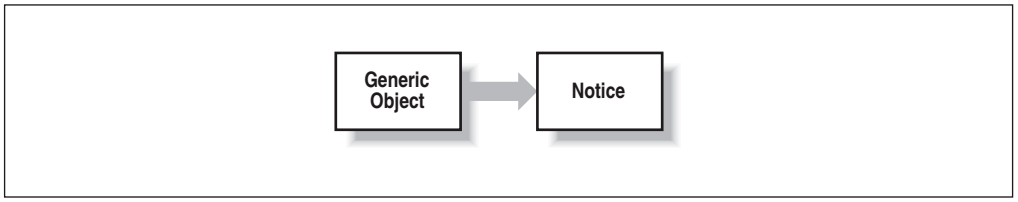

Figure 12-1. Notice class hierarchy

XView defines two types of notices, *Standard notices* and *screen-locking* notices:

- *Standard notices* do not lock the screen and are placed centered in the "owner" frame. This type of notice may either block the application's thread of execution, or not block.
- *Screen-locking notices* lock the screen and block input to all applications (the screen is locked with X grabs). These notices appear with a shadow that emanates from the location where an action in an application initiates the notice. This may be a panel button, such as "Quit," or some other XView object.

New applications that are created with XView Version 3 should use the `NOTICE` package described in this chapter. Older versions of XView only supported notices with a nonobject-oriented interface using the `notice_prompt()` function. For compatibility, `notice_prompt()` is still supported. However, for new applications its use is not recommended. Furthermore, the `NOTICE` package is implemented so that updating applications to use a notice object is an easy task. Notice objects are only created when the notice package is used (they are not created when `notice_prompt()` is used). For more information on `notice_prompt()`, refer to Appendix B, *Notices*.

Figure 12-2 shows an example of a notice window from the *OPEN LOOK GUI Specification*.

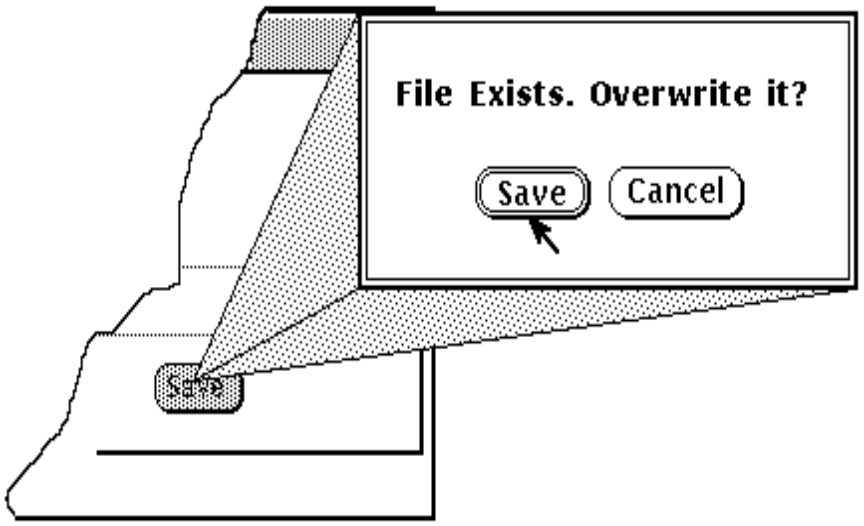

Figure 12-2. A sample notice window

12.1 Creating and Displaying Notices

To use the `NOTICE` package, include the header file *<xview/notice.h>*. It provides the necessary types and definitions for using the package. A notice object's type is `Xv_Notice`. In general, create a notice like any other XView object:

```
Xv_Notice   notice;
Frame       owner;
notice = xv_create(owner, NOTICE,
            NOTICE_MESSAGE_STRING,
                 "Please confirm your action.",
                 NULL,
            NULL);
```

Make a notice visible by setting `XV_SHOW` to `TRUE`:

```
xv_set(notice, XV_SHOW, TRUE, NULL);
```

Clicking on any button in a notice pops-down the notice. It is *not* necessary to set `XV_SHOW` to `FALSE` on a notice (the notice package handles this internally).

A notice must have an owner that is a subtype of `WINDOW`, such as a frame or a panel. If `NULL` is used for the owner, an error results and the notice is not created. Typically the window of the application that causes the notice to be created is the notice's owner. For example, 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.

Your application has control over the type of the notice (standard or screen-locking), the messages that are displayed in the notice window, and the choices available to the user as responses. The notice package creates the notice window, and depending on the type of the notice, either blocks input to the originating application and does *not* lock the screen while waiting for the user to make a selection on one of the available button choices, or, locks the screen and waits for one of the button choices to be selected (in this case, the screen is frozen and the application is blocked.) For both types of notices, after the user enters a response in a notice button, the notice window is unmapped.

The attributes NOTICE_MESSAGE_STRING, NOTICE_MESSAGE_STRINGS, and NOTICE_MESSAGE_STRINGS_ARRAY_PTR can be used to determine what strings are to be displayed in a notice. NOTICE_MESSAGE_STRING takes as its value one NULL-terminated string. This string, however, can contain the character "\n" to serve as a line break. The attribute NOTICE_MESSAGE_STRINGS can take a list of the above strings as its value. This list is NULL-terminated. Each string in the list will start on a new line. Lastly, NOTICE_MESSAGE_STRINGS_ARRAY_PTR takes a pointer to a NULL-terminated array of strings.

All the strings mentioned above are centered horizontally in the notice.

Below is an example using NOTICE_MESSAGE_STRING:

```
Xv_Notice   notice;
Frame       owner;
notice = xv_create(owner, NOTICE,
          NOTICE_MESSAGE_STRING,
            "Hello!\nPlease confirm your action.\nPress Continue",
          NULL);
```

The following example demonstrates NOTICE_MESSAGE_STRINGS:

```
Xv_Notice   notice;
Frame       owner;
notice = xv_create(owner, NOTICE,
            NOTICE_MESSAGE_STRINGS,
                 "Hello!",
                 "Please confirm your action.\nPress Continue",
            NULL,
            NULL);
```

NOTICE_MESSAGE_STRINGS_ARRAY_PTR takes an array of strings as in the following example:

```
char        array[5];

array[0]    = "Hello!";
array[1]    = "This is a sample notice.";
array[2]    = "Press Continue.";
array[3]    = NULL;

Xv_Notice   notice;
Frame       owner;

notice = xv_create(owner, NOTICE,
            NOTICE_MESSAGE_STRINGS_ARRAY_PTR,
                 array,
            NULL);
```

A sample standard notice is demonstrated in Example 12-1.

Example 12-1. 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>

Panel       panel;

main(argc,argv)
  int     argc;
  char    *argv[ ];
{
    Frame       frame;
    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);
}

/*
 * my_notify_proc() -- called when the user selects the QUIT button.
 *      Here the user must choose YES or NO on the notice to confirm
 *      or deny quitting.
 */
Xv_opaque
my_notify_proc(item, event)
  Panel_item  item;
  Event      *event;
{
    Xv_notice   notice;
    int         notice_stat;

    notice = xv_create(panel, NOTICE,
        NOTICE_MESSAGE_STRINGS, "Do you really want to quit?", NULL,
        NOTICE_BUTTON_YES,      "Yes",
        NOTICE_BUTTON_NO,       "No",
        NOTICE_STATUS,          &notice_stat,
```

Example 12-1. The simple_notice.c program (continued)

```
        XV_SHOW, TRUE,
        NULL);

    switch (notice_stat)  {
        case NOTICE_YES:
     /* Quit */
            exit(0);
        break;

        case NOTICE_NO:
     /* Don't quit */
        break;

    }

    xv_destroy_safe(notice);
}
```

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 12-3. If the user presses "Yes," the program exits.

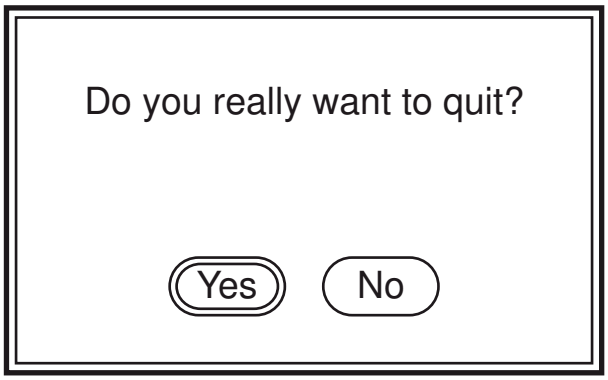

Figure 12-3. Output of simple_notice.c while the notice is up

12.1.1 Notice Values and Status

Two responses are normally available 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 values correspond to the notice's status. The notice status contains the value of the button that was selected on the notice. The location where the status is stored can be set with the attribute NOTICE_STATUS. If NOTICE_STATUS is not set, the notice status can still be obtained by using xv_get() on NOTICE_STATUS. If the attributes NOTICE_BUTTON_YES and NOTICE_BUTTON_NO are used, the notice status is either set to NOTICE_YES or NOTICE_NO, depending upon the user's notice button selection, which causes the notice to pop-down. As shown in *simple_notice.c*, the strings associated with the "Yes" and "No" buttons are set with the attributes NOTICE_BUTTON_YES and NOTICE_BUTTON_NO. The notice button created with NOTICE_BUTTON_YES is the default button and will have the OPEN LOOK default ring around it.

The notice choices listed above also respond to accelerator keys. In other words, in addition to the RETURN key, whatever key that is mapped to the semantic action ACTION_DEFAULT_ACTION can also be used to select the NOTICE_BUTTON_YES button. The key that is mapped to the semantic action ACTION_STOP can be used to select the NOTICE_BUTTON_NO button.

When the notice window is mapped, the cursor is immediately warped (moved) to the default button since it is the default response of the notice.

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 more than two choices, use the NOTICE_BUTTON attribute to define the choices available:

```
xv_create(panel, NOTICE,
    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 value for the button selection. The application should make its decision on how to proceed based on the button value specified. In this case, the notice would need to handle cases for NOTICE_STATUS having values of 101, 102, and 103 (in addition to possible errors).

Because the NOTICE_BUTTON attribute is used, there is no button that is bound to the default NOTICE_YES choice. But since OPEN LOOK requires a default button for every notice, the default button will be the first button. In this case, the "Quit, Update changes" button will have the default ring around it.

If no buttons are specified for a notice, a default button labeled "Confirm" with a value set to NOTICE_YES is provided.

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

The Mouseless Model allows keyboard actions for selecting and moving between notice buttons (refer to Section 6.13, “The Mouseless Model,” in Chapter 6, *Handling Input*).

12.2 Types of Notices

The `simple_notice.c` program shows a default, standard notice. The default notice does not lock the screen but does block the thread of execution. This section describes the two types of notices:

- Standard notices that do not lock the screen. Standard notices have two varieties: those that block the input to the application, and those that do not.
- Screen-locking notices that lock the screen and blocks input to the application.

The `Boolean` attribute `NOTICE_LOCK_SCREEN` determines the type of the notice. When `NOTICE_LOCK_SCREEN` is `FALSE`, the attribute `NOTICE_BLOCK_THREAD` determines whether the notice blocks the input to the application.

After creating a notice and popping it up or down, you can change its type. For example, if `notice` is created with `NOTICE_LOCK_SCREEN` set to `TRUE` (a screen-locking notice), the following call is valid:

```
xv_set(notice, NOTICE_LOCK_SCREEN, FALSE,
     NULL);
```

After doing this your notice is a standard notice and you can, if necessary, use the type-specific attributes that apply to the standard notice (such as `NOTICE_BLOCK_THREAD`).

12.2.1 Standard Notices

When the attribute `NOTICE_LOCK_SCREEN` is set to `FALSE`, the notice is a *standard notice* and it does not lock the screen (this is the default value for `NOTICE_LOCK_SCREEN`). Whether it blocks the thread of execution depends upon the value of the attribute `NOTICE_BLOCK_THREAD` (its default value is `TRUE`). The attribute `NOTICE_STATUS` is used to determine which button was pressed.

When the notice is displayed, the state of the application’s other windows is as follows:

- If `NOTICE_BLOCK_THREAD` is set to `TRUE`, then no windows of the application, except the notice window, will detect mouse and keyboard input (also see `xv_window_loop()`).
- If `NOTICE_BLOCK_THREAD` is set to `FALSE`, only the frame that owns the notice will ignore mouse and keyboard input. Additional frames that need to be put in this state can be added with `NOTICE_BUSY_FRAMES`. All such frames will have their headers grayed out (also see `FRAME_BUSY`).

Table 12-1 lists additional notice attributes that apply only to standard notices (when NOTICE_LOCK_SCREEN is FALSE).

Table 12-1. Notice Attributes (used with NOTICE_LOCK_SCREEN = FALSE)

Attribute	Procedures
NOTICE_BLOCK_THREAD	create, set
NOTICE_BUSY_FRAMES	create, set
NOTICE_EVENT_PROC	create, set

Standard notices are centered in the owner frame and are always on top of the frame that owns them. If the frame is in the iconified state when the notice is mapped, the notice will be placed centered at the location of the pointer.

NOTICE_BLOCK_THREAD is relevant only for standard notices, those that have NOTICE_LOCK_SCREEN = FALSE. Example 12-2 shows a standard, thread-blocking notice.

Example 12-2. Creating a standard notice

```
/*
 * Display a notice that does not lock the screen.
 * This doesn't return until a button on the notice is pressed.
 * This is a standard blocking notice.
 */
frame_notice = xv_create(frame, NOTICE,
            NOTICE_LOCK_SCREEN, FALSE, /* default */
            NOTICE_BLOCK_THREAD, TRUE, /* default */
            NOTICE_MESSAGE_STRINGS,
                "Are you sure you want to Quit?",
            NULL,
            NOTICE_BUTTON_YES, "Confirm",
            NOTICE_BUTTON_NO, "Cancel",
            NOTICE_NO_BEEPING, TRUE,
            NOTICE_STATUS, &result,
            XV_SHOW, TRUE,
            NULL);

switch (result)  {
case NOTICE_YES:
     /*confirm */
      exit(0);
      break;
case NOTICE_NO:
   /* Cancel */
      break;
default:
      break;
}
```

12.2.1.1 Using a notice callback

The previous notice examples did not use a callback for notice events. A callback may be defined to handle the notice events for a standard notice. The attribute NOTICE_EVENT_PROC specifies an application-defined callback procedure that is called when any of the buttons on the notice are selected. This procedure has the following format:

```
void
notice_event_proc(notice, value, event)
    Xv_Notice  notice;    /* public handle to notice      */
    int        value;     /* value associated with button */
    Event      *event;    /* Pointer to struct with event info.*/
```

This procedure is called before the notice pops down. If a procedure is not specified for NOTICE_EVENT_PROC, the notice still pops down when a button is pressed. Example 12-3 demonstrates a notice using NOTICE_EVENT_PROC. In this example, the callback procedure my_notice_event_proc() is defined to handle all the notice button selections.

Example 12-3. A notice using a callback

```
xv_create(parent, NOTICE,
        NOTICE_LOCK_SCREEN, FALSE,
        NOTICE_BUTTON, "Save Changes", 100,
        NOTICE_BUTTON, "Cancel", 101,
        NOTICE_BUTTON, "Quit", 102,
        NOTICE_MESSAGE_STRINGS,
            "Press Save Changes to save changes to file and quit",
            "Press Cancel to continue",
            "Press Quit to quit",
        NULL,
        NOTICE_EVENT_PROC, my_notice_event_proc,
        NULL);


/* my_notice_event_proc() Procedure */

my_notice_event_proc(notice, value, event)
Xv_Notice  notice;
int        value;
Event      *event;
{
    switch(value)  {

    case 100:
        /* code for save changes */
    break;

    case 101:
        /* code for cancel */
    break;

    case 102:
        /* code for quit */
    break;

    default:
        printf("Bad button value!!\n");
```

Example 12-3. A notice using a callback (continued)

```
    break;
    }

}
```

12.2.1.2 Selecting the busy frames

You can use NOTICE_BUSY_FRAMES to specify the frames or sub-windows that should be set to "busy" during notice pop-up. NOTICE_BUSY_FRAMES only takes frames for its values. The following code shows how to use NOTICE_BUSY_FRAMES.

```
void   my_notice_event_proc();

xv_create(parent, NOTICE,
        NOTICE_LOCK_SCREEN, FALSE,
        NOTICE_BUTTON, "Save Changes", 100,
        NOTICE_BUTTON, "Cancel", 101,
        NOTICE_BUTTON, "Quit", 102,
        NOTICE_MESSAGE_STRINGS,
            "Press Save Changes to save changes to file and quit",
            "Press Cancel to continue",
            "Press Quit to quit",
        NULL,
        NOTICE_EVENT_PROC, my_notice_event_proc,
          NOTICE_BUSY_FRAMES,
               sub_frame1,  /* frames to make busy  */
               sub_frame2,  /* during pop-up.       */
               NULL,
        NULL);
```

12.2.2 Notices That Lock the Screen

To create a screen-locking notice, set NOTICE_LOCK_SCREEN to TRUE. Screen-locking notices lock the screen *and* block the thread of execution for the application. You create a screen-locking notice as follows:

```
notice = xv_create(frame, NOTICE,
            NOTICE_LOCK_SCREEN, TRUE,
            NOTICE_MESSAGE_STRINGS,
                "Are you sure you want to Quit?",
                NULL,
            NOTICE_BUTTON_YES, "Confirm",
            NOTICE_BUTTON_NO, "Cancel",
            XV_SHOW, TRUE,
            NULL);
```

Similarly, the calls below set the type of the notice, `error_notice`, to make it a screen-locking notice.

```
xv_set(error_notice, XV_SHOW, TRUE,
     NOTICE_LOCK_SCREEN, TRUE,
     NULL);
```

All screen-locking notices block the thread of execution. Additional attributes apply when `NOTICE_LOCK_SCREEN` is `TRUE`. Table 12-2 lists the attributes that apply when `NOTICE_LOCK_SCREEN` is `TRUE`.

Table 12-2. Screen-Locking Notice Attributes (for NOTICE_LOCK_SCREEN = TRUE)

Attribute	Procedures
NOTICE_FOCUS_XY	create, set
NOTICE_TRIGGER	create, set
NOTICE_TRIGGER_EVENT	create, set

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 maps the notice. Example 12-4 shows code for a screen-locking notice.

Example 12-4. Creating a screen-locking notice

```
int  return_val;
/* Create notice
 * Pop up notice with shadow from (100, 200) relative to "parent"
 * - this blocks
 */
notice = xv_create(parent, NOTICE,
     NOTICE_LOCK_SCREEN, TRUE,
     NOTICE_BUTTON, "Save Changes", 100,
     NOTICE_BUTTON, "Cancel", 101,
     NOTICE_BUTTON, "Quit", 102,
     NOTICE_MESSAGE_STRINGS,
        "Press Save Changes to save changes to file and quit",
        "Press Cancel to continue",
        "Press Quit to quit",
     NULL,
     NOTICE_FOCUS_XY, 100, 200,
     NOTICE_STATUS, &notice_stat,
     XV_SHOW, TRUE,
     NULL);

/*
* Notice pops down when a button is pressed.
* notice_stat contains the value of the button that was clicked on.
*/
switch (notice_stat)  {
         case 100:
                /* save changes */
              break;
```

Example 12-4. Creating a screen-locking notice (continued)

```
        case 101:
             /* cancel */
            break;

        case 102:
             /* quit */
            break;
}
```

12.2.2.1 Notice triggers

If you want to assign accelerators to screen-locking 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 pop down the notice. The value of `NOTICE_STATUS` in this case is `NOTICE_TRIGGERED`, and the event that caused the trigger will be in the `Event` specified by `NOTICE_TRIGGER_EVENT`. When triggers are not used, the `Event` pointer can be `NULL`. Example 12-5 shows how to use `NOTICE_TRIGGER` to catch a particular event in a notice.

Example 12-5. 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 */
```

Example 12-5. The trigger_notice.c program (continued)

```
    window_fit(panel);
    window_fit(frame);

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

/*
 * my_notify_proc() -- called when the user selects the "Move"
 * panel button.  Put up a notice 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);
    Xv_notice   notice;

    x = event_x(event), y = event_y(event);
    printf("original click relative to panel: %d, %d0, x, y);
    notice = xv_create(panel, NOTICE,
     NOTICE_LOCK_SCREEN, TRUE,
     NOTICE_TRIGGER_EVENT, event,
     NOTICE_STATUS, &result,
     XV_SHOW, TRUE,
        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);
    }

    xv_destroy_safe(notice);
}
```

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.

When the notice pops down, the `Event` structure that `NOTICE_TRIGGER_EVENT` points to contains the event that triggered the notice (popped it down). 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 in `NOTICE_TRIGGER_EVENT` when the notice pops down, 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.

12.3 Destroying a Notice

Notices can be destroyed with `xv_destroy(notice)`. If a notice is destroyed when it is visible, it will be taken down. Use `xv_destroy_safe()` if the destruction is done in a `NOTICE_EVENT_PROC`.

12.4 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 12-6 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. In previous examples, the notice is destroyed immediately after it is unmapped and the status is obtained. In this example, it is not destroyed, but is reused over and over again.

Example 12-6. 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
```

Example 12-6. The notice.c program (continued)

```
 * 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();

    /*
     * 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
         * the notice.
         */
        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
     */
```

Example 12-6. The notice.c program (continued)

```
    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
         * notices
         */
        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 "key data" associated with the panel item is extracted via
 * 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;
    static Xv_notice    notice = NULL;

    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.
     */
    if (!notice)  {
        notice = xv_create(panel, NOTICE,
          NOTICE_LOCK_SCREEN, TRUE,
          NOTICE_STATUS, &result,
          XV_SHOW, TRUE,
            NOTICE_MESSAGE_STRINGS,
                    prompt,
                    "Press YES to confirm",
                    "Press NO to cancel",
                    NULL,
```

Example 12-6. The notice.c program (continued)

```
            NOTICE_BUTTON_YES,      "YES",
            NOTICE_BUTTON_NO,       "NO",
            NULL);
    }
    else  {
      /*
       * If the notice has been created, just set its
       * message strings and show it.
       */
      xv_set(notice,
          XV_SHOW, TRUE,
            NOTICE_MESSAGE_STRINGS,
                    prompt,
                    "Press YES to confirm",
                    "Press NO to cancel",
                    NULL,
          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);
    }
}
```


12.5 Notice Package Summary

Table 12-3 lists the attributes for the `NOTICE` package. These attributes are described fully in the *XView Reference Manual*.

Table 12-3. Notice Attributes

NOTICE_BLOCK_THREAD	NOTICE_LOCK_SCREEN
NOTICE_BUSY_FRAMES	NOTICE_MESSAGE_STRING
NOTICE_BUTTON	NOTICE_MESSAGE_STRINGS
NOTICE_BUTTON_NO	NOTICE_MESSAGE_STRINGS_ARRAY_PTR
NOTICE_BUTTON_YES	NOTICE_NO_BEEPING
NOTICE_EVENT_PROC	NOTICE_STATUS
NOTICE_FOCUS_XY	NOTICE_TRIGGER
NOTICE_FONT	NOTICE_TRIGGER_EVENT
XV_KEY_DATA	XV_SHOW