14
Icons

A user may *close* an application to save space on the display. The program is still running and it may even be active, but it is not receiving input from the user. In order to represent the application in its closed state, an *icon* is used. An icon is a small picture that represents the application, as shown in Figure 14-1 from the *OPEN LOOK GUI Specification Guide*.

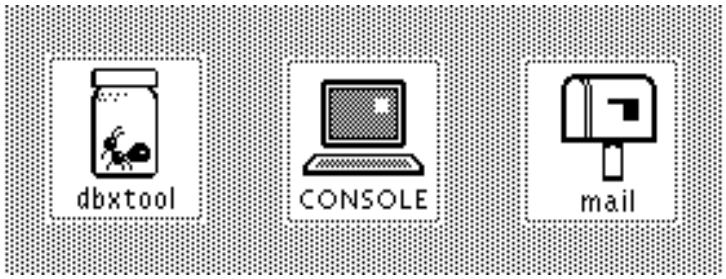

Figure 14-1. Three bordered default icons

The graphic image that icons use may be used for other purposes and, therefore, may be shared among other objects in the application. But the icon image should be designed to easily identify the application while in a closed state. Icons may also have text associated with them. Space is limited, so the text is usually the name of the application.

14.1 Creating and Destroying Icons

To use the `ICON` package, include the header file *<xview/icon.h>*. Figure 14-2 shows the class hierarchy for the `ICON` package.

The form for creating an icon is:

```
Icon icon;

icon = (Icon)xv_create(owner, ICON, attributes, NULL);
```

The `owner` of an icon is a base frame, but it may be created with a `NULL` owner. Once an icon is assigned to a frame, the owner of the icon is changed to that frame. This is another example of *delayed binding*.

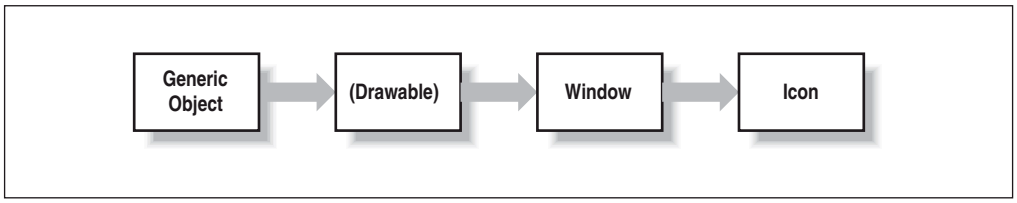

Figure 14-2. Icon package class hierarchy

When destroying an icon, the server image associated with the icon is not destroyed—it is your responsibility to free the server image and the pixmap associated with the icon if needed.

14.2 The Icon's Image

The most important thing about the icon is its graphic representation, so you will also need to be familiar with the SERVER_IMAGE package described in Chapter 15, *Nonvisual Objects*. Once an image is created, you can create an icon and assign it to a frame. This chapter does not discuss the creation of server images for icons, whether they originate from filenames or from the actual data.

The program in Example 14-1 creates two server images—it uses *open.icon* as the image for the panel button and *closed.icon* for the application's icon.* Pressing the panel button causes the application to *close* to its iconic state. You must use a window manager function to open the application back up again.

Example 14-1. The icon_demo.c program

```
/*
 * icon_demo.c -- demonstrate how an icon is used.  Create a server
 * image and create an icon object with the image as the ICON_IMAGE.
 * Use the icon as the frame's icon.
 */
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/svrimage.h>
#include <xview/icon.h>

short open_bits[ ] =  {
#include "open.icon"
};

short closed_bits[ ] =  {
#include "closed.icon"
};

main(argc, argv)
```

*The files *open.icon* and *closed.icon* are not included in this book due to their length and complexity. They are bitmap files represented in ASCII *hex* notation. The files are included with the XView distribution.

Example 14-1. The icon_demo.c program (continued)

```
int     argc;
char    *argv[ ];
{
    Frame               frame;
    Panel               panel;
    Server_image        open_image, closed_image;
    Icon                icon;
    void                close_frame();

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

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

    open_image = (Server_image)xv_create(NULL, SERVER_IMAGE,
        XV_WIDTH,               64,
        XV_HEIGHT,              64,
        SERVER_IMAGE_BITS,      open_bits,
        NULL);

    closed_image = (Server_image)xv_create(NULL, SERVER_IMAGE,
        XV_WIDTH,               64,
        XV_HEIGHT,              64,
        SERVER_IMAGE_BITS,      closed_bits,
        NULL);

    (void) xv_create(panel, PANEL_MESSAGE,
        PANEL_LABEL_IMAGE,      open_image,
        PANEL_NOTIFY_PROC,      close_frame,
        NULL);

    icon = (Icon)xv_create(frame, ICON,
        ICON_IMAGE,             closed_image,
        XV_X,                   100,
        XV_Y,                   100,
        NULL);
    xv_set(frame, FRAME_ICON, icon, NULL);

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

void
close_frame(item, event)
Panel_item item;
Event *event;
{
    Frame frame = (Frame)xv_get(xv_get(item,
        PANEL_PARENT_PANEL), XV_OWNER);
    xv_set(frame, FRAME_CLOSED, TRUE, NULL);
}
```

The callback routine for the panel button, `close_frame()`, makes a call to an XView routine which sends window manager requests from the client to the window manager. In this case, we set the frame's `FRAME_CLOSED` attribute to `TRUE` to request that the window manager iconify the application associated with the `frame` parameter.*

The position of the image with respect to the icon is set using the attribute `ICON_IMAGE_RECT`, which takes as its value a pointer to a `Rect` structure. The `r_top` and `r_left` fields of the structure indicate the offset from the upperleft corner of the icon where the image is placed. The `r_width` and `r_height` fields describe the size of the image. If the icon is going to be a different size from the size of the icon's image, or if there is going to be text used with this icon, then the `ICON_IMAGE_RECT` attribute should be used. Section 14.2.1, "The Icon Text," has an example.

14.2.0.1 Color icons

You can make color icons in several ways. You can create a color server image and use that as the `ICON_IMAGE`. You can set the foreground and background colors for a monochrome image (1-bit deep image) and change the colormap of the icon. For example:

```
Icon          icon;
unsigned long foreground_index, background_index;
Server_image  image;  /* assume 1-bit deep monochrome image */
Cms           cms;

icon = (Icon)xv_create(frame, ICON,
    ICON_IMAGE,           image,
    WIN_CMS,              cms,
    WIN_FOREGROUND_COLOR, foreground_index,
    WIN_BACKGROUND_COLOR, background_index,
    NULL);
```

The icon created assumes that the colormap object, cms, has been created. The foreground and background colors are indices into the colormap (see Chapter 21, *Color*, for more information on colormap segments). Also see the program *x_draw.c* in Appendix F, *Example Programs*.

Example 14-2. Color cursors

```
#include <xview/xview.h>
#include <xview/svrimage.h>
#include <xview/cms.h>

/* Icon data */
static unsigned short icon_bits[ ] = {
#include "cardback.icon"
};

main(argc,argv)
int     argc;
char    *argv[ ];
{
```

*Window manager functions are discussed in Chapter 4, *Frames*.

Example 14-2. Color cursors (continued)

```
    Frame         frame;

    Icon          icon;
    unsigned long foreground_index, background_index;
    Server_image  image;  /* assume 1-bit deep monochrome image */
    Cms           cms;

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

    frame = xv_create(NULL,FRAME,NULL);

    image = (Server_image)xv_create(NULL, SERVER_IMAGE,
        XV_WIDTH,             64,
        XV_HEIGHT,            64,
        SERVER_IMAGE_DEPTH,   1,
        SERVER_IMAGE_BITS,    icon_bits,
        NULL);

    foreground_index = 2;
    background_index = 0;

    cms = (Cms)xv_create(NULL,CMS,
        CMS_SIZE,             3,
        CMS_NAMED_COLORS,     "green","red","blue",NULL,
        NULL);

    icon = (Icon)xv_create(frame, ICON,
        ICON_IMAGE,           image,
        WIN_CMS,              cms,
        WIN_FOREGROUND_COLOR, foreground_index,
        WIN_BACKGROUND_COLOR, background_index,
        NULL);

    xv_set(frame, FRAME_ICON, icon, NULL);

    xv_main_loop(frame);
}
```

14.2.0.2 ICON_TRANSPARENT

If the `ICON_TRANSPARENT` attribute is set to `TRUE` (it's `FALSE` by default), the icon's background color is set to the background color of the root window. This may give the effect that the icon is transparent in the case where the root window is a solid color. However, be careful when the root window has a backing bitmap pattern or a different colormap from the icon.

14.2.0.3 ICON_MASK_IMAGE

The attribute `ICON_MASK_IMAGE` may be used to clip all drawing into the pixmap to the bits set in the `Pixmap` or `Server_image` specified here. Therefore, this image should be a 1-bit deep bitmap. The image used as the icon mask is usually a "shadow" of the icon's normal image. That is, it is the same image "filled in," resulting in a totally black 1-bit deep icon that has the same shape as the icon's image.

Example 14-2 can be modified to use this attribute by adding the following code:

```
image_mask = (Server_image)xv_create(NULL, SERVER_IMAGE,
    XV_WIDTH,            64,
    XV_HEIGHT,           64,
    XV_DEPTH,            1,
    SERVER_IMAGE_BITS,   closed_image_mask_bits,
    NULL);

icon = (Icon)xv_create(frame, ICON,
    ICON_IMAGE,      closed_image,
    ICON_MASK_IMAGE, image_mask,
    XV_X,            100,
    XV_Y,            100,
    NULL);
```

When used in conjunction with the `ICON_TRANSPARENT` attribute, it may be possible to create an icon that appears to have a shape other than a square.

14.2.1 The Icon Text

Each icon can have text associated with it. This text is not part of the icon's image; it is rendered on top of the image after the image is rendered. To specify the text displayed in the icon, use the generic attribute `ICON_LABEL`.* By default, the text is displayed at the bottom of the icon area. This may overlap the icon's image. You can change the position in which the text is rendered by using the attribute `ICON_LABEL_RECT`. The value of the attribute describes a rectangular region in which the text will overwrite anything underneath it. If the text does not fit, it is clipped by this region. To keep the entire image on the icon and display the text without writing over the image, define your icon to be large enough to include both the image and the extents of the text without these regions overlapping. This might make your icon a nonstandard size, however. (The size of an icon is typically 64x64 pixels.)

The code fragment in Example 14-3 implements added sizes to compensate for text while preserving enough area to display the entire icon.

Example 14-3. Redefining an icon's size to include its label

```
...
Rect         image_rect, label_rect;
Server_image image;
Icon         icon;
...
```

*`ICON_LABEL` is defined to be `XV_LABEL` in <*xview/icon.h*>.

Example 14-3. Redefining an icon's size to include its label (continued)

```
rect_construct(&image_rect, 0, 20, 64, 64);
rect_construct(&label_rect, 0, 0, 64, 20);
icon = xv_create(frame, ICON,
    XV_WIDTH,               64,
    XV_HEIGHT,              64 + 20,
    XV_LABEL,               "Sample",
    ICON_LABEL_RECT,        &label_rect,
    ICON_IMAGE,             image,
    ICON_IMAGE_RECT,        &image_rect,
    NULL);
xv_set(frame, FRAME_ICON, icon, NULL);
```

The first thing we do is construct the image and label area by using the `rect_construct()` macro found in *<xview/rect.h>*. The image is positioned at `0, 20` and is a size of 64x64. The text is positioned at the upperleft corner (`0, 0`), extends to the width of the icon and is 20 pixels high. The size of the icon is 64, and the height of the text is 20, so when we create the icon, we set the height of the icon object using `XV_HEIGHT` at `64+20`.

14.2.2 ICON_TRANSPARENT_LABEL

The `ICON_TRANSPARENT_LABEL` attribute specifies a string that is drawn into the icon using the foreground color only. Pixels other than those in the font set are not affected.

Creating, setting, and getting `ICON_TRANSPARENT_LABEL` is equivalent to creating, setting, and getting `ICON_LABEL`, except that the string is drawn in the foreground color only.

14.3 Icon Package Summary

There are no procedures or macros in the `ICON` package. Table 14-1 shows the attributes for the `ICON` package. This information is described fully in the *XView Reference Manual*.

Table 14-1. Icon Attributes

`ICON_FONT`	`ICON_LABEL_RECT`
`ICON_HEIGHT`	`ICON_MASK_IMAGE`
`ICON_IMAGE`	`ICON_TRANSPARENT`
`ICON_IMAGE_RECT`	`ICON_TRANSPARENT_LABEL`
`ICON_LABEL`	`ICON_WIDTH`
`XV_LABEL`	