21
Color

The X Window System has various ways of allocating, specifying, and using colors. While all of these methods are available to applications without XView intervening, XView provides its own model for color specification that may be used as an alternative. It does not provide anything more than what is already available, but it may provide a simpler interface to request and specify colors. This model is especially useful when specifying colors for XView objects, such as panel buttons and scrollbars.

This chapter does not directly discuss how to use colormaps and related Xlib color-specific functions. For a discussion of them, see Volume One, *Xlib Programming Manual*, Chapter 7, *Color*. This chapter discusses only the XView color model.

Obviously, the user cannot view colors in an application without having a color display. But you cannot tell at the time your application is written whether the user's display is going to be able to support color. You can use the `DisplayDepth()` macro to determine whether the user's display can handle color:

```
Display *dpy = (Display *)xv_get(frame, XV_DISPLAY);
extern use_color;

if (DefaultDepth(dpy, DefaultScreen(dpy)) < 2)
    use_color = False;
```

21.1 XView Color Model

XView applications deal with color by using *colormap segments*. Use the `CMS` package to create a colormap segment. Figure 21-1 shows the class hierarchy for the `CMS` package.

As a simple introduction, the following code fragment creates a colormap segment with the specified colors and returns a handle to it:

```
cms = (Cms)xv_create(NULL, CMS,
    CMS_SIZE,         4,
    CMS_NAMED_COLORS, "white", "red", "green", "blue", NULL,
    NULL);
```

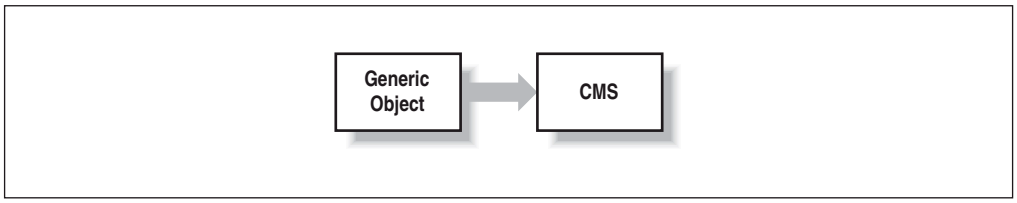

Figure 21-1. CMS package class hierarchy

Window-based objects (canvases, panels, textsw, etc.) use colormap segments to get their colors. These objects get a default colormap segment when they are created, but you can assign a new one using the `WIN_CMS` attribute:

```
canvas = (Canvas)xv_create(frame, CANVAS,
    WIN_CMS, cms,
    NULL);
```

Colormap segments must be applied to windows to assure that the window can access the color you are attempting to draw into. However, there is much to understand about colors, colormaps, colormap segments, visuals, servers, and X11 to be able to use colors in an efficient and robust way. Unless done correctly, you may produce code that only works on specific machines or in specific environments.

21.1.0.1 What is a colormap segment?

A colormap segment (cms), is a subset of the available cells in a colormap on the X server. These are XView entities (i.e., not Xlib) that provide a veneer over the Xlib color mechanism. Colormap segments can be created as either *static* or *dynamic* and are derived from an underlying colormap of the same type.

Any object subclassed from the window object may allocate and use colormap segments. You can use Xlib routines to do all your color and colormap manipulation within canvas windows, pixmaps, and other X-related objects, but you must use the colormap segment API for XView objects.

More than one XView object may reference the same colormap segment. However, a colormap segment does not keep track of the objects that are using it. The application is required to keep track of changes in colors and update its objects accordingly.

The internals to XView attempt to create colormap segments that are as small as possible so that numerous segments can share the same underlying colormap. If a colormap segment requires more colors than the current colormap has space for, a new colormap must be created.

21.1.1 Colormap Segment Types

You can create static or dynamic colormap segments. The type is set with the `CMS_TYPE` attribute. Its value can be either `XV_STATIC_CMS` or `XV_DYNAMIC_CMS`. A colormap segment's type cannot change, so the `CMS_TYPE` attribute is for `xv_create()` only.

The X11 Protocol specifies that the `Visual` type of a window must be declared at the time it is created. Therefore, XView objects that allocate colormap segments are required, at creation time, to specify the type of visual they propose to use. The default visual is determined by the default visual of the screen.

21.1.1.1 Static colormap segments

Applications must always use static colormap segments unless they require read-write colors. Colors allocated from a static colormap segment are shared among *all* applications. In static colormap segments, when a new color is asked for, the XView library will try to return the closest (or exact) matching color from the server (using `XAllocColor()`) if the default colormap on the server is `StaticColor`.

Whenever possible, a colormap segment is derived from the default colormap obtained from the screen in which the window resides (e.g., `DefaultColormap()`). Only when the colors on that colormap have been exhausted is a new colormap allocated. If XView needs to allocate a new colormap for a new cms, it does this internally. It is impossible for the application to specify a colormap for a colormap segment.

The cells in a static cms are initialized once and are read-only from then on. Static colormap segments, by sharing color cells across applications, use the shared hardware colormap resources more efficiently and reduce *flashing*. Flashing is a blinking effect you sometimes get when moving the cursor in and out of various windows on the screen. This is caused by the server popping different colormaps in and out as you move from one window to the next.

21.1.1.2 Dynamic colormap segments

When you ask for a dynamic cms, XView sends a request to the server to allocate read-write colors. When colors are requested from this cms, the color returned is the exact color; the closest match is not returned as it is with static colors.

21.2 Creating Colormap Segments

Applications that use color must include the file *<xview/cms.h>*. A cms can be created using the standard call to `xv_create()` with the package name `CMS`.

```
Cms cms;

cms = (Cms)xv_create(parent, CMS, attrs, NULL);
```

The *parent* of a colormap segment is the XView screen object with which the colormap is associated. If a parent is not specified, the default screen of the default server is used as the parent.

21.2.0.1 Cms size

A cms may contain as many colors as you like as long as they fit within the largest colormap you can create. Having more than one colormap segment reference the same color value is perfectly legal and reasonable. Data is frequently shared among segments for optimal efficiency.

When creating a colormap segment, you must specify its size (i.e., the number of colors it has) using the `CMS_SIZE` attribute. If you don't set the size, it defaults to the macro, `XV_DEFAULT_CMS_SIZE`, which is `2`. `CMS_SIZE` is a create-only and get-only attribute; once a colormap is created, its size cannot be changed, although you can query a colormap's size using `xv_get()`.

If all the colors in a colormap segment are not initialized, the uninitialized colors are undefined and should not be used. You can change colors within a dynamic cms at any time. Uninitialized colors of a static cms can be initialized using `xv_set()`, but once initialized, they may not be changed.

You might want to create a segment of a larger size than the number of colors you assign it because you may not have all the colors you know you'll need right away. For example, you create a colormap segment of size *n*, but initialize it with only *n-4* colors. When the rest of the colors are ready to be loaded into the segment, you do so at the location of the uninitialized colors, namely, at index *n-4*. The `CMS_INDEX` attribute is used to specify this location.

CAUTION

> All the colors in a static cms must be specified at the time of creation to avoid race conditions in the associated X11 colormap. Because static colormaps are shared with other applications, if you request *n* colors but do not initialize all of them, another application could request and initialize enough colors to fill up the colormap before you get a chance to set the rest of your colors.

When creating a cms or setting new colors, you may specify `CMS_COLOR_COUNT` to indicate the number of colors to load. Again, if you want to load these colors at a position other than the beginning of the segment, use `CMS_INDEX`. This is typically used only when you are creating a colormap segment and not initializing each color right away. Or, if you have

already done this, you are adding more colors to a prebuilt colormap segment that hasn't had all of its colors initialized. Therefore, this is a create-only and set-only attribute.

21.2.1 Specifying Colors

You can specify actual colors by name or by RGB values. When using RGB values, you can use XView or Xlib data structures. In each case, we are going to create a colormap segment with the same four colors: white, red, green, and blue.

21.2.1.1 Specifying colors by name

The attribute CMS_NAMED_COLORS takes as its value a NULL-terminated list of strings representing color names:

```
cms = (Cms)xv_create(parent, CMS,
    CMS_SIZE,         4,
    CMS_NAMED_COLORS, "white", "red", "green", "blue", NULL,
    NULL);
```

The colors specified by the names are converted into actual values using XParseColor()* and allocated into the colormap segment using XAllocColor() (for static colormaps) or XStoreColor() (for dynamic colormaps). The example shown probably works because the colors used are common colors found on most X servers' color databases. However, you should be careful when requesting named colors in this fashion because the database may not contain the color name you specify. If any of the colors requested fails, then no change to the cms is affected and xv_set() returns XV_ERROR. If a cms is being created via xv_create() and an error occurs, no cms is created and xv_create() returns NULL.

CMS_NAMED_COLORS cannot be used by xv_get().

21.2.1.2 Specifying colors by RGB values

You can request colors more directly by specifying the actual red, green, and blue (RGB) values using one of two attributes. CMS_COLORS takes as a value an array of Xv_singlecolor objects. This XView-defined type is declared as:

```
typedef struct xv_singlecolor {
    unsigned char red, green, blue;
} Xv_singlecolor;
```

*XParseColor() is an Xlib call that maps char * color names into RGB values.

We can use the following to produce a cms with the same colors:

```
static Xv_singlecolor colors[ ] = {
    { 255, 255, 255, }, /* white */
    { 255, 0, 0 },      /* red */
    { 0, 255 0, },      /* green */
    { 0, 0, 255 },      /* blue */
};
cms = xv_create(NULL, CMS,
    CMS_SIZE,    4,
    CMS_COLORS,  colors,
    NULL);
```

Alternatively, you can use the attribute `CMS_X_COLORS` to specify an array of `XColor` structures, defined in *<X11/Xlib.h>* as:

```
typedef struct {
    unsigned       long pixel;
    unsigned short red, green, blue;
    char           flags;  /* do_red, do_green, do_blue */
    char           pad;
} XColor;
```

Here is a way to produce a colormap segment with the same colors but using an array of `XColors`:

```
static XColor colors[ ] = {
/* white */ { 0, 255<<8, 255<<8, 255<<8, DoRed|DoGreen|DoBlue, 0 },
/* red */   { 0, 255<<8,      0,      0, DoRed|DoGreen|DoBlue, 0 },
/* green */ { 0,    0,   255<<8,      0, DoRed|DoGreen|DoBlue, 0 },
/* blue */  { 0,    0,        0, 255<<8, DoRed|DoGreen|DoBlue, 0 },
};
cms = xv_create(NULL, CMS,
    CMS_SIZE,      4,
    CMS_X_COLORS,  colors,
    NULL);
```

Note that the color values in the red, green, and blue fields of the `XColor` data structure are left-shifted by 8. For more details on specifying colors with Xlib, see Volume One, *Xlib Programming Manual*.

When storing colors, if the colormap segment type is static, XView uses `XAllocColor()`. If the colormap segment type is dynamic, XView uses `XAllocColorCells()` and `XStoreColors()` to allocate a dynamic colormap and store the requested colors in it.

After setting colors in a cms, the pixel values for the colors can be retrieved. These pixel values are indices into the colormap itself, not the colormap *segment*. See Section 21.3, “Color and Pixel Values.”

The XView attributes to set colors can be used to get the colors from the cms as well. For example, to get the colors into an array of `Xv_singlecolor`, use:

```
Xv_singlecolor colors[SIZE];

xv_get(cms, CMS_COLORS, colors);
```

The size of the `colors` array *must be the same size as the color segment* because `xv_get()` gets the entire colormap segment and stores the colors at the base of the array.

You cannot get a partial list of colors from the cms. The same is true for getting the colors into an array of `XColor`:

```
XColor colors[SIZE];

xv_get(cms, CMS_X_COLORS, &colors);
```

In each case, `xv_get()` returns a pointer to the base of the array of data structures passed as the third argument. We choose to ignore it here because the array itself is sufficient.

21.2.2 Cms Name

Colormap segments can be named using `CMS_NAME`.* Windows can change between colormaps by setting their `WIN_CMS_NAME` to the names of allocated colormap segments. However, this old-style method of specifying colormap segments for windows is made obsolete by the attribute `WIN_CMS`, the recommended method for assigning colormap segments to windows.

If unnamed, a cms will get a unique name assigned to it. You can retrieve that name using `xv_get()` and `CMS_NAME`. `CMS_NAME` is also the only attribute for which you can use `xv_find()` for the `CMS` package.

21.3 Color and Pixel Values

By now, you know that a *color* is defined to be a set of red, green, and blue (RGB) intensity values. For example, red is represented by a full intensity for the red value and no intensity for the green and blue values. Other shades are achieved by raising or lowering the intensities of one or more of the RGB values. A colormap is an array of RGB values; a *pixel* is an index into that array.

21.3.0.1 Logical versus real indices

If a colormap segment of size *n* is created, its logical indices range from 0 to *n*-1. XView attributes that take color values (e.g., `WIN_FOREGROUND_COLOR`, `PANEL_ITEM_COLOR`, etc.) always deal with logical index values.

The *real* indices of a colormap segment are the actual indices into the hardware colormap. Each colormap segment maintains an internal table to translate from logical to real indices. Real index values (*pixels*) are used for setting the foreground and background colors in `GC`s used in Xlib calls.

*`CMS_NAME` is defined to be `XV_NAME`.

The index table from a colormap segment can be obtained using the attribute CMS_INDEX_TABLE.

```
unsigned long *colors;

colors = (unsigned long *)xv_get(cms, CMS_INDEX_TABLE);
```

Similarly, the pixel values from a *window-based object* can be obtained using the window attribute, WIN_X_COLOR_INDICES:

```
unsigned long *colors;

colors = (unsigned long *)xv_get(canvas, WIN_X_COLOR_INDICES);
```

Note the object passed to xv_get().

In both cases, the returned value is a pointer to an array of unsigned long types. For a colormap segment of size *n*, colors[0], colors[1], . . . colors[*n*-1] contain the actual pixel values corresponding to the colors in the underlying X11 colormap. These pixel values can be used in calls to XSetForeground() or XSetBackground() to change GC values.

To get the real pixel value corresponding to a logical index value, you can use the CMS_PIXEL attribute:

```
Cms cms;
unsigned long red, blue;

cms = (Cms)xv_create(NULL, CMS,
    CMS_SIZE,          2,
    CMS_NAMED_COLORS,  "red", "blue", NULL,
    NULL);

red = (unsigned long)xv_get(cms, CMS_PIXEL, 0);
blue = (unsigned long)xv_get(cms, CMS_PIXEL, 1);
```

CMS_PIXEL can only be used to get the value of pixels; you cannot use it to set cms color values.

As discussed earlier, the CMS_X_COLORS attribute can be used to get an array of XColor elements. This data structure has a pixel field that can be referenced to get the pixel values associated with colors.

21.3.1 Foreground and Background Colors

Foreground and background colors correspond to the last and first colors in a colormap segment. That is, the segment's *background* color corresponds to the logical index 0 in the cms, whereas the *foreground* color corresponds to the logical index *n*-1 (where *n* is the size of the cms).

Identifying the foreground and background colors on certain XView objects may not be as straightforward as it appears. For example, the background color of a canvas is the color the entire canvas is painted with when XClearArea() is called. The foreground color is a thin border color around the inside perimeter of each canvas view window. This is not

necessarily the color in which graphics are rendered into the canvas using `XCopyArea()` or `XDrawString()`. The color of the graphical images you see in a canvas is dependent on the foreground color set in the `GC` used by the Xlib routines.

The real pixel values for the foreground and background colors of a cms may be obtained directly using the attributes `CMS_FOREGROUND_PIXEL` and `CMS_BACKGROUND_PIXEL`.

```
Cms            cms;
unsigned long  fg, bg;

bg = (unsigned long)xv_get(cms, CMS_BACKGROUND_PIXEL);
fg = (unsigned long)xv_get(cms, CMS_FOREGROUND_PIXEL);
```

21.3.1.1 Colors of control objects

A *control object* in XView refers to panels, scrollbars, notices, and menus. These packages cannot have their foreground and background modified programmatically in the same way as other window-based objects. This restriction applies to the 3D interface; the 2D interface may allow the foreground and background colors to be modified. However, OPEN LOOK states that the background of all control objects appear consistent; thus, a single *control color* is used. On panels, it is possible, but not recommended, to set the `WIN_FOREGROUND_COLOR` and `WIN_BACKGROUND_COLOR` attributes. Panel items, may have their colors set by setting `PANEL_ITEM_COLOR` to a logical index into the panel's cms. XView allows the user, not the programmer, to set the background color for control objects via the resource database. This is discussed in Section 21.5, "The Control Colormap Segment."

21.4 The color_logo.c Program

Using the basic principles discussed so far, we present Example 21-1 to demonstrate the creation and initialization of a colormap segment for an XView canvas. Pixel values are extracted from the colormap segment and set into a GC's foreground. Xlib calls are then used to render various items in the same four basic colors we've been using.

Example 21-1. The color_logo.c program

```
/* color_logo.c --
 *  This program demonstrates the combined use of the XView color
 *  model/API and Xlib graphics calls. The program uses XView to
 *  create and manage its colormap segment while doing its actual
 *  drawing using Xlib routines.
 *  The program draws the X logo in red, green and blue in a canvas.
 */
#include <xview/xview.h>
#include <xview/canvas.h>
#include <xview/cms.h>
#include <xview/xv_xrect.h>
#include <X11/bitmaps/xlogo64>

/* Color indices */
```

Example 21-1. The color_logo.c program (continued)

```
#define WHITE           0
#define RED             1
#define GREEN           2
#define BLUE            3
#define NUM_COLORS      4

GC gc;                      /* used for rendering logos */
unsigned long *pixel_table; /* pixel values for colors */
Pixmap        xlogo;        /* the xlogo */

/* Create a frame, canvas, and a colormap segment and assign the
 * cms to the canvas.  CMS_INDEX_TABLE returns the actual colormap
 * indices and are used to set the gc's foreground for XCopyPlane
 * calls.
 */
main(argc,argv)
int     argc;
char    *argv[ ];
{
    Frame         frame;
    XGCValues     gc_val;
    XGCValues     gcvalues;
    void          canvas_repaint_proc();
    Cms           cms;
    static Xv_singlecolor colors[ ] = {
        { 255, 255, 255 }, /* white */
        { 255,   0,   0 }, /* red */
        { 0,   255,   0 }, /* green */
        { 0,     0, 255 }, /* blue */
    };

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

    cms = (Cms) xv_create(NULL, CMS,
        CMS_SIZE, 4,
        CMS_COLORS, colors,
        NULL);

    frame = (Frame)xv_create(XV_NULL, FRAME,
        FRAME_LABEL,    argv[0],
        XV_WIDTH,       448,
        XV_HEIGHT,      192,
        NULL);

    (void) xv_create(frame, CANVAS,
        CANVAS_X_PAINT_WINDOW,  TRUE,
        CANVAS_REPAINT_PROC,    canvas_repaint_proc,
        WIN_CMS,                cms,
        NULL);

    /* Get the actual indices into the colormap */
    pixel_table = (unsigned long *)xv_get(cms, CMS_INDEX_TABLE);

    /* create the xlogo -- get display/window from the frame obj */
    xlogo = XCreateBitmapFromData(
        xv_get(frame, XV_DISPLAY), xv_get(frame, XV_XID),
```

Example 21-1. The color_logo.c program (continued)

```
        xlogo64_bits, xlogo64_width, xlogo64_height);

    /* setup gc for rendering logos to screen */
    gcvalues.graphics_exposures = False;
    gcvalues.background = pixel_table[WHITE];
    gc = XCreateGC(xv_get(frame, XV_DISPLAY), xv_get(frame, XV_XID),
        GCBackground | GCGraphicsExposures, &gcvalues);

    xv_main_loop(frame);
}

/* Draws onto the canvas using Xlib drawing functions.
 * Draw the X logo into the window in three colors. In each case,
 * change the GC's foreground color to the pixel value specified.
 */
void
canvas_repaint_proc(canvas, pw, display, win, xrects)
Canvas      canvas;   /* unused */
Xv_Window   pw;       /* unused */
Display     *display;
Window      win;
Xv_xrectlist *xrects; /* unused */
{
    /* Use XCopyPlane because the logo is a 1-bit deep pixmap. */
    XSetForeground(display, gc, pixel_table[RED]);
    XCopyPlane(display, xlogo, win, gc, 0, 0,
        xlogo64_width, xlogo64_height, 64, 64, 1);

    XSetForeground(display, gc, pixel_table[GREEN]);
    XCopyPlane(display, xlogo, win, gc, 0, 0,
        xlogo64_width, xlogo64_height, 192, 64, 1);

    XSetForeground(display, gc, pixel_table[BLUE]);
    XCopyPlane(display, xlogo, win, gc, 0, 0,
        xlogo64_width, xlogo64_height, 320, 64, 1);
}
```

Example 21-1 uses Xlib routines to draw into the canvas's paint window. Therefore, the GC's foreground color is set to an index from the colormap being used by that paint window. In order to get the correct color from the colormap, we need to get the color table for the window using the attribute CMS_COLOR_INDEX. The repaint routine draws the X logo in the specified colors.

21.5 The Control Colormap Segment

The management of colormap segments is a little different for control objects (panels, notices, menus, etc.) than it is for other XView objects. In order for XView to provide the same control colors for control objects, these objects must use a *control colormap segment*. This is just like a normal cms except that parts of it are reserved for the predefined colors.

Since OPEN LOOK suggests that the background of control objects in an application appear in a consistent color, XView sets that color to be that specified by the resource `OpenWindows.WindowColor` from the user's environment. This color is used as the background color, and along with a few others, they are set aside in the first few indices of the control colormap segment.

These *control* colors are used to provide a 3D look for the control objects. Thus, the control colormap segment must be used for all control objects.

Aside from the added colors, there is little difference between a control cms and a normal cms. The cms may still request many colors and there is no limit to the choice of colors used in the new cms. However, these extra colors cannot be used as the background for control objects.

A control colormap segment is created by setting the boolean attribute `CMS_CONTROL_CMS` to `TRUE` in the call to `xv_create()`. The macro `CMS_CONTROL_COLORS` (defined in <*xview/cms.h*>) indicates how many predefined control colors there are, so the first `CMS_CONTROL_COLORS` indices in the cms are initialized by the XView library. If the application requires *n* other colors in this cms, it must explicitly ask that the segment be created with a size of:

```
n + CMS_CONTROL_COLORS
```

In such a case, the application must refer to its own *n* colors using the index range:

```
CMS_CONTROL_COLORS to CMS_CONTROL_COLORS + n-1
```

An application-defined colormap segment set on a control object (such as a panel) must be a control colormap segment so that the object can be pointed with the 3D look.

```
#define WHITE       0
#define RED         1
#define GREEN       2
#define BLUE        3
#define NUM_COLORS  4

control_cms = xv_create(NULL, CMS,
    CMS_SIZE,          CMS_CONTROL_COLORS + NUM_COLORS,
    CMS_CONTROL_CMS,   TRUE,
    CMS_NAMED_COLORS,  "white", "red", "green", "blue", NULL,
    NULL);
```

We set the boolean attribute CMS_CONTROL_CMS to TRUE to indicate that we are creating a control cms. Notice that the size of the colormap segment is the number of colors *we* specified plus the number of control colors. This colormap segment contains both control colors and our colors; XView automatically allocates our colors after the control colors.

When we create the panel, we specify the new colormap segment as the panel's cms using the common window attribute, WIN_CMS:

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

When we reference our own specified colors, we must offset those color values by CMS_CONTROL_COLORS:

```
/* assume a 1-bit deep 16x16 square pixmap */
extern Server_image chip;

xv_create(panel, PANEL_CHOICE,
    PANEL_LABEL_STRING, "Colors",
    PANEL_CHOICE_IMAGES, chip, chip, chip, chip, NULL,
    PANEL_CHOICE_COLOR, 0, WHITE + CMS_CONTROL_COLORS,
    PANEL_CHOICE_COLOR, 1, RED + CMS_CONTROL_COLORS,
    PANEL_CHOICE_COLOR, 2, GREEN + CMS_CONTROL_COLORS,
    PANEL_CHOICE_COLOR, 3, BLUE + CMS_CONTROL_COLORS,
    NULL);
```

Here, we create a choice item whose choices are colored "chips" in solid colors corresponding to the color names offset by the control colormap segment.

21.5.1 Coloring Panel Items

You can specify colors for panel items using PANEL_ITEM_COLOR. This attribute takes as a value an index into a colormap segment. The value -1 is reserved for the panel's foreground color, whatever that may be. It is also the default color of panel items unless you have specified otherwise with PANEL_ITEM_COLOR. The scrollbar on a PANEL_LIST will always take on the window color that is specified in the Workspace Properties sheet. PANEL_ITEM_COLOR on a PANEL_LIST will only affect the scrolling list's label, title, and rows. Remember, if you're going to be using the 3D interface, then you must create the cms as a *control* cms. Example 21-2 briefly demonstrates how to create colored panel items.

Example 21-2. The color_panel.c program

```
/* color_panel.c --
 * This program demonstrates how to set panel items to different
 * colors using the XView API for color.
 */
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/cms.h>

/* Color indices */
#define WHITE           0
#define RED             1
```

Example 21-2. The color_panel.c program (continued)

```
#define GREEN           2
#define BLUE            3
#define NUM_COLORS      4

/* Create a frame, panel, and a colormap segment and assign the
 * cms to the panel.
 */
main(argc,argv)
int     argc;
char    *argv[ ];
{
    Frame       frame;
    Panel       panel;
    Cms         cms;
    extern void exit(), pressed();
    static Xv_singlecolor colors[ ] = {
        { 255, 255, 255 }, /* white */
        { 255,   0,   0 }, /* red */
        { 0,   255,   0 }, /* green */
        { 0,     0, 255 }, /* blue */
    };

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

    cms = (Cms) xv_create(NULL, CMS,
        CMS_CONTROL_CMS,        TRUE,
        CMS_SIZE,               CMS_CONTROL_COLORS + 4,
        CMS_COLORS,             colors,
        NULL);

    frame = (Frame)xv_create(XV_NULL, FRAME,
        FRAME_LABEL,            argv[0],
        FRAME_SHOW_FOOTER,      TRUE,
        NULL);

    panel = xv_create(frame, PANEL,
        WIN_CMS,        cms,
        NULL);

    xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Red",
        PANEL_ITEM_COLOR,       CMS_CONTROL_COLORS + RED,
        PANEL_NOTIFY_PROC,      pressed,
        NULL);
    xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Green",
        PANEL_ITEM_COLOR,       CMS_CONTROL_COLORS + GREEN,
        PANEL_NOTIFY_PROC,      pressed,
        NULL);
    xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Blue",
        PANEL_ITEM_COLOR,       CMS_CONTROL_COLORS + BLUE,
        PANEL_NOTIFY_PROC,      pressed,
        NULL);
    xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Quit",
```

Example 21-2. The color_panel.c program (continued)

```
        PANEL_ITEM_COLOR,       CMS_CONTROL_COLORS + WHITE,
        PANEL_NOTIFY_PROC,      exit,
        NULL);

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

void
pressed(item, event)
Panel_item item;
Event *event;
{
    char *name = (char *)xv_get(item, PANEL_LABEL_STRING);
    Frame frame = xv_get(xv_get(item, PANEL_PARENT_PANEL), XV_OWNER);

    xv_set(frame, FRAME_LEFT_FOOTER, name, NULL);
}
```

Notice how the color index for each panel item is offset by the number of colors in the control color item. If all the references to the CMS_CONTROL_COLORS and CMS_CONTROL_CMS attributes were removed, the result would be a 2D panel whose panel items are the same colors as their names.

21.6 Using xv_find() with Colormap Segments

xv_find() can be used to find a previously created colormap segment. Currently, the attribute CMS_NAME is the only attribute used by xv_find() to find a match.

```
cms = (Cms)xv_find(screen, CMS,
    CMS_NAME, "palette",
    XV_AUTO_CREATE, FALSE,
    NULL);
```

This example returns a handle to a colormap segment whose name is "palette." XV_AUTO_CREATE is set to FALSE to prevent a new colormap segment from being created. If the colormap segment with that name is not found, then NULL is returned.

21.7 Canvases and Colormaps

When the colormap segment associated with a canvas is changed, the contents of the canvas must be repainted to reflect the new colors. If the boolean attribute `CANVAS_CMS_REPAINT` is set to `TRUE`, the library automatically calls the canvas's repaint procedure each time a new colormap segment is set on the canvas. The damage list passed to the routine contains the dimensions of the entire paint window.

Note that the application itself must track any changes in the contents of a colormap segment. `CANVAS_CMS_REPAINT` enables the library to generate a synthetic repaint event only when the actual colormap segment is switched.

For dynamic colormap segments, when a color changes, the pixel value remains the same but the *color* represented by the index into the colormap segment changes. Therefore, the repaint routine is not called and the window's appearance changes automatically. This method of colormap manipulation is commonly used to implement color animation.

21.8 Multi-visual Support

XView allows you to create windows and colormap segments using arbitrary visuals (See Chapter 7 of Volume One, *Xlib Programming Manual* for a description of visuals in X11). You can use this functionality to access any visual that the X11 server supports. The attributes described in this section allow you to indicate visuals you want associated with your windows and colormap segments.

The attribute `XV_VISUAL` specifies the exact visual that will be used in the creation of a window or colormap segment. The value is a pointer a `Visual` structure (available from `XMatchVisualInfo` or `XGetVisualInfo`). This attribute applies only to `WINDOW` and `CMS` objects.

The attribute `XV_VISUAL_CLASS` specifies the class of the visual that will be used in the creation of the window or colormap segment. The value should be one of the following:

```
StaticGray
GrayScale
StaticColor
PseudoColor
TrueColor
DirectColor
```

If the server that the window is being created for does not support the visual class specified, XView uses the default visual. `XV_VISUAL_CLASS` also only applies to a `WINDOW` or to a `CMS` object.

The attribute `WIN_DEPTH` specifies the depth of the window that you are creating. If the value specified by this attribute is not supported by the server, the library uses the default depth instead. This attribute, like all `WIN_*` attributes applies only to `WINDOW` objects.

21.8.1 Using the Visual Attributes

Both XV_VISUAL and XV_VISUAL_CLASS do essentially the same thing. XV_VISUAL is more specific. Use XV_VISUAL_CLASS, optionally in conjunction with WIN_DEPTH, if you have no preference about the specific visual, but just require a certain visual class. However, if you want to make sure that you are using a specific visual that the server supports, use XGetVisualInfo or XMatchVisualInfo to find the visual and then use the visual as an argument to XV_VISUAL.

If you are creating your own colormap segment(s), make sure that they are created with the same visual as the window you will be using with the CMS. For example, if you have a window that requires a cms, do the following:

```
cms = (Cms)xv_create(screen, CMS,
                     XV_VISUAL, (Visual *)xv_get(window, XV_VISUAL),
                     ...
                     NULL);
```

Note that you should never assume that creating a window with XV_VISUAL_CLASS or WIN_DEPTH will use the class and depth you specify. If the server does not have a visual of the given class and depth, you will not be supplied with the visual you specify. If your application depends on using a specific visual, be sure to check the visual class and depth of the window, using xv_get(), to see if you did indeed get the required visual. Alternatively, you may query the server before the creating the window, using XGetVisualInfo or XMatchVisualInfo, to check the visuals that the server supports.

21.9 Another Example

This final example demonstrates just about all of the features discussed in this chapter. It includes creating a colormap segment, initializing color, using foreground and background colors, and setting colors on specific XView objects, including panel items.

In Example 21-3, the user selects objects (from the “Objects” item) to be colored by selecting one of the color tiles from the “Colors” choice item. The callback function for this panel item calls color_notify(), which sets the currently selected colors on the foreground or background of the selected objects. Whether to use foreground or background colors is dependent on the value of the “Fg/Bg” panel item; the items whose colors are affected are retrieved by getting the value of the “Objects” panel item. Since more than one object can be selected from this item, the value is a mask of the selected items. We loop through each bit in the mask identifying which objects should have their colors set.

Example 21-3. The color_objs.c program

```
/*
 * color_objs.c --
 *    This program demonstrates the use of color in XView. It allows
 *    the user to choose the foreground and background colors of the
 *    various objects in an interactive manner.
 */
#include <xview/xview.h>
```

Example 21-3. The color_objs.c program (continued)

```
#include <xview/svrimage.h>
#include <xview/textsw.h>
#include <xview/panel.h>
#include <xview/cms.h>
#include <xview/notice.h>

#define SELECT_TEXTSW          0
#define SELECT_TEXTSW_VIEW     1
#define SELECT_PANEL           2
#define SELECT_ICON            3

#define NUM_COLORS             8

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

/* solid black square */
static short black_bits[ ] = {
    0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
    0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF
};

Panel_item      objects;
Textsw          textsw;
Panel           panel;
Icon            icon;

/*
 * main()
 *    Create a panel and panel items. The application uses panel items
 *    to choose a particular object and change its foreground and
 *    background colors in an interactive manner. Create a textsw.
 *    Create an icon. All the objects share the same colormap segment.
 */
main(argc,argv)
int     argc;
char    *argv[ ];
{
    Frame         frame;
    Panel_item    color_choices, panel_fg_bg;
    Cms           cms;
    int           i;
    Server_image  chip, icon_image;
    void          color_notify();
    extern void   exit();
    static Xv_singlecolor cms_colors[ ] = {
        { 255, 255, 255 }, /* white  */
        { 255, 0, 0 },     /* red    */
        { 0, 255, 0 },     /* green  */
        { 0, 0, 255 },     /* blue   */
        { 255, 255, 0 },   /* yellow */
        { 188, 143, 143 }, /* brown  */
        { 220, 220, 220 }, /* gray   */
        { 0, 0, 0 },       /* black  */
```

Example 21-3. The color_objs.c program (continued)

```
};

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

frame = (Frame)xv_create(NULL, FRAME,
    FRAME_LABEL,    argv[0],
    NULL);

cms = (Cms)xv_create(NULL, CMS,
    CMS_NAME,           "palette",
    CMS_CONTROL_CMS,    TRUE,
    CMS_TYPE,           XV_STATIC_CMS,
    CMS_SIZE,           CMS_CONTROL_COLORS + NUM_COLORS,
    CMS_COLORS,         cms_colors,
    NULL);

/* Create panel and set the colormap segment on the panel */
panel = (Panel)xv_create(frame, PANEL,
    PANEL_LAYOUT,       PANEL_VERTICAL,
    WIN_CMS,            cms,
    NULL);

/* Create panel items */
objects = (Panel_item)xv_create(panel, PANEL_TOGGLE,
    PANEL_LABEL_STRING,     "Objects",
    PANEL_LAYOUT,           PANEL_HORIZONTAL,
    PANEL_CHOICE_STRINGS,   "Textsw", "Textsw View",
                            "Panel", "Icon", NULL,
    NULL);

panel_fg_bg = (Panel_item)xv_create(panel, PANEL_CHECK_BOX,
    PANEL_LABEL_STRING,     "Fg/Bg",
    PANEL_CHOOSE_ONE,       TRUE,
    PANEL_LAYOUT,           PANEL_HORIZONTAL,
    PANEL_CHOICE_STRINGS,   "Background", "Foreground", NULL,
    NULL);

chip = (Server_image)xv_create(XV_NULL, SERVER_IMAGE,
    XV_WIDTH,           16,
    XV_HEIGHT,          16,
    SERVER_IMAGE_DEPTH, 1,
    SERVER_IMAGE_BITS,  black_bits,
    NULL);
color_choices = (Panel_item)xv_create(panel, PANEL_CHOICE,
    PANEL_LAYOUT,           PANEL_HORIZONTAL,
    PANEL_LABEL_STRING,     "Colors",
    PANEL_CLIENT_DATA,      panel_fg_bg,
    XV_X,                   (int)xv_get(panel_fg_bg, XV_X),
    PANEL_NEXT_ROW,         15,
    PANEL_CHOICE_IMAGES,
        chip, chip, chip, chip, chip, chip, chip, chip, NULL,
    PANEL_CHOICE_COLOR,     0, CMS_CONTROL_COLORS + 0,
    PANEL_CHOICE_COLOR,     1, CMS_CONTROL_COLORS + 1,
    PANEL_CHOICE_COLOR,     2, CMS_CONTROL_COLORS + 2,
    PANEL_CHOICE_COLOR,     3, CMS_CONTROL_COLORS + 3,
    PANEL_CHOICE_COLOR,     4, CMS_CONTROL_COLORS + 4,
```

Example 21-3. The color_objs.c program (continued)

```
        PANEL_CHOICE_COLOR,     5, CMS_CONTROL_COLORS + 5,
        PANEL_CHOICE_COLOR,     6, CMS_CONTROL_COLORS + 6,
        PANEL_CHOICE_COLOR,     7, CMS_CONTROL_COLORS + 7,
        PANEL_NOTIFY_PROC,      color_notify,
        NULL);

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

    /* create textsw and set the colormap segment for it */
    textsw = (Textsw)xv_create(frame, TEXTSW,
        WIN_CMS,              cms,
        WIN_BELOW,            panel,
        WIN_ROWS,             15,
        WIN_COLUMNS,          80,
        TEXTSW_FILE_CONTENTS, "/etc/motd",
        WIN_BACKGROUND_COLOR, CMS_CONTROL_COLORS + 0,
        NULL);

    /* adjust panel dimensions */
    (void)xv_set(panel, WIN_WIDTH, xv_get(textsw, WIN_WIDTH), NULL);

    icon_image = (Server_image)xv_create(NULL, SERVER_IMAGE,
        XV_WIDTH,             64,
        XV_HEIGHT,            64,
        SERVER_IMAGE_DEPTH,   1,
        SERVER_IMAGE_BITS,    icon_bits,
        NULL);
    /* associate icon with the base frame */
    icon = (Icon)xv_create(XV_NULL, ICON,
        ICON_IMAGE,           icon_image,
        WIN_CMS,              cms,
        WIN_BACKGROUND_COLOR, CMS_CONTROL_COLORS + 0,
        NULL);
    xv_set(frame, FRAME_ICON, icon, NULL);

    window_fit(frame);

    xv_main_loop(frame);
}

/*
 * This routine gets called when a color selection is made.
 * Set the foreground or background on the currently selected object.
 * WIN_FOREGROUND_COLOR & WIN_BACKGROUND_COLOR allow the application
 * to specify indices into the colormap segment as the foreground
 * and background values.
 */
void
color_notify(panel_item, choice, event)
Panel_item      panel_item;
int             choice;
Event           *event;
```

Example 21-3. The color_objs.c program (continued)

```
{
    int cnt;
    Xv_opaque object, get_object();
    unsigned objs = (unsigned)xv_get(objects, PANEL_VALUE);
    int fg = (int)xv_get(xv_get(panel_item, PANEL_CLIENT_DATA),
                    PANEL_VALUE);

    /* the value of the objects panel item is a bit mask ... "on" bits
     * mean that the choice is selected.  Get the object associated
     * with the choice and set it's color.  "&" tests bits in a mask.
     */
    for (cnt = 0; objs; cnt++, objs >>= 1)
        if (objs & 1)
            if ((object = get_object(cnt)) != panel)
                xv_set(object,
                    fg? WIN_FOREGROUND_COLOR : WIN_BACKGROUND_COLOR,
                    CMS_CONTROL_COLORS + choice, NULL);
            else if (fg)
                PANEL_EACH_ITEM(panel, panel_item)
                    xv_set(panel_item,
                        PANEL_ITEM_COLOR, CMS_CONTROL_COLORS + choice,
                        NULL);
                PANEL_END_EACH
            else
                notice_prompt(panel, NULL,
                    NOTICE_FOCUS_XY, event_x(event), event_y(event),
                    NOTICE_MESSAGE_STRINGS,
                        "You can't set the color of a panel.", NULL,
                    NOTICE_BUTTON_YES, "Ok",
                    NULL);
}

/*
 *    Return the XView handle to nth object.
 */
Xv_opaque
get_object(n)
int n;
{
    switch (n) {
        case SELECT_TEXTSW:
            return textsw;
        case SELECT_TEXTSW_VIEW:
            return xv_get(textsw, OPENWIN_NTH_VIEW, 0);
        case SELECT_PANEL:
            return panel;
        case SELECT_ICON:
            return icon;
        default:
            return textsw;
    }
}
```

21.10 Cms Package Summary

Table 21-1 lists the attributes for the `CMS` package. This information is described fully in the *XView Reference Manual*.

Table 21-1. Cms Attributes

CMS_BACKGROUND_PIXEL	CMS_NAME
CMS_COLOR_COUNT	CMS_NAMED_COLOR
CMS_COLORS	CMS_PIXEL
CMS_CONTROL_CMS	CMS_SCREEN
CMS_FOREGROUND_PIXEL	CMS_SIZE
CMS_INDEX	CMS_TYPE
CMS_INDEX_TABLE	CMS_X_COLORS
XV_DEPTH	XV_VISUAL
XV_VISUAL_CLASS	