14

Specifying Color

In the X Window System, color can be a complicated business. However, it doesn't have to be; you can keep it simple. Every X-based system comes with a pre-fabricated database of colors to choose from. You can access these colors by naming them in an option on the command line or as a resource in a resource file. The clients and the server work out the rest.

Most of the color names you can think of, plus many more esoteric names, are included in the standard color database. So odds are you can just pick a name such as "dark blue" or "violet" and the database will deliver. A case in point: the following command creates an *xbiff* window with a dark blue foreground (for the mailbox picture) and an elegant violet background.

```
xbiff -bg violet -fg "dark blue" &
```

This chapter describes the traditional X methods of using color to decorate your X windows. If you are using color inside an application (e.g., you are using the GIMP for doing graphics), this technology still applies, but you'll also want to look at the documentation for the software you are using and perhaps read Chapter 3, *Selection of Useful X Clients*. The X desktops, GNOME and KDE, have their own tools for managing color—see Chapter 8, *Using GNOME*, or Chapter 9, *Using KDE*, for information on configuring color in your GNOME or KDE desktop.

Using Color

A color display uses multiple bits per pixel to determine what color to display for that pixel. The number of bits associated with each pixel is referred to as the *depth* of the display.[*] The depth determines how many colors are available; that is, how many colors

[*] You'll also see depth referred to as the number of *planes* or *bitplanes*. Think of a plane running through the first bit of each pixel on your screen, another plane running through the second bit of each pixel, etc. Thus a display with a depth of 8 has 8 planes; if the depth is 16, there are 16 planes.

can be displayed at the same time. Old monochrome displays used just one bit per pixel, resulting in two possible colors (e.g., black and white, or black and amber) depending on whether the bit was on or off. Modern color display hardware normally has a color depth of at least 16 bits, giving a possible 65,536 (2^{16}) colors, while a depth of 24 results in 16,777,216 (2^{24}) color choices and a depth of 32 allows for over 4 billion (2^{32}) colors.

The color depth is determined for an X session when the server is started. The number of available color depths is a function of your monitor's capabilities and the amount of memory on your video card. When you installed X, the configuration process determined the valid depths for your X server.[*] If you look at *XF86Config*, you'll see the valid choices for your X server. Once the server is running at a particular depth, all users who log in will run at that depth.

If you start an X session manually with *startx,* you can specify the depth on the command line for that session, overriding the default value in *XF86Config*:

```
startx -- -depth 16
```

The *–depth 16* option specifies the depth of the display.[1] The X server then has 65,536 different colors available for use. Unfortunately, knowing how many colors are available doesn't always tell you *which* colors you can use. We'll talk more about why this is true in the section "How Colors Work".

The rest of this section talks briefly about some of the things you can do with color; we'll go into more detail about how you use color and about the technology, in the remainder of the chapter.

Color as Command-line Options and Resources

All X applications written using an Xt-based toolkit, as mentioned in Chapter 12, *Setting Resources*, let you specify foreground and background colors using either command-line options or resource variables. Specifically, these options and resources are: *–fg, –bg*, `foreground,` and `background`. There is also a *–bd* option, which lets you set a border color. However, it's not particularly useful since most window managers provide their own decorations that largely supersede the client's border.

In addition to letting you set the foreground and background colors, many clients have additional resources that let you set colors for particular features of an application. For example, *xclock* provides options and resources to specify colors for the hands of the clock (*–hd*, `hands`) and the edges of the hands (*–hl*, `highlight`). You can combine these with the standard background and foreground options/resources for the clock face (background) and the tick marks (foreground) to set up a very colorful clock. Similarly, the *xterm* command allows you to set colors for the text cursor (*–cr*, `cursorColor`) and the pointer symbol (*–ms*, `pointerColor`, `pointerColorBackground`) as well the foreground text color and the background for the window itself. If you are interested in

[*] Chapter 10, *XFree86*, explains in detail the configuration of the X server.

[1] If you are using an older version of XFree86, use the *–bpp* option instead, to specify the depth as the number of bits per pixel: *-bpp 16*.

setting colors for an X client, see the client's documentation or manpage to find out what color options and resources you can set.

Perhaps the most dramatic use of color is as a root-window background. The *xsetroot* client allows you to specify a solid color for the root window. For example, the following command sets the root window color:

```
xsetroot -solid steelblue
```

This command can be placed in an X initialization file so it executes at logon time.

But you aren't limited to a solid color. If you want something fancier, you can set the root window to a bitmap, which you can make as colorful as you'd like. Here's an example using the *xsnow* bitmap that comes with X:

```
xsetroot -fg lightblue -bg darkblue \
         -bitmap /usr/X11R6/include/X11/bitmaps/xsnow
```

This command fills the root window with light-blue snowflakes tiled on a darker blue background. See Chapter 3 and the *xsetroot* manpage for information on creating your own bitmaps and using them as root-window patterns.

In addition, your window manager allows color customization, letting you specify colors for window manager decorations, menus, dialog boxes, etc. For many window managers, predefined themes have been created and are available on web sites such as *http://www.themes.org/*. A theme establishes a color scheme that provides a consistent look for all windows on your desktop.

Finding Colors

How do you find out the names of the colors? The standard colors are listed in the text file */usr/X11R6/lib/X11/rgb.txt*. You can examine the file just as you would any other text file, and pick colors that sound appealing. The section "The RGB Color Model" later in this chapter talks more about *rgb.txt* and its use.

Another method is to use the *showrgb* client. When you run *showrgb*, it appears to do nothing more than *cat* the file to your terminal window. But *showrgb* has the advantage that you don't have to remember the full pathname to *rgb.txt*. Given the number of colors, you'll want to pipe the command's output to a paging program, such as *more*, or pipe the output to *grep*. For example, to find all the RGB colors that have orange in their name, you could use the following command:

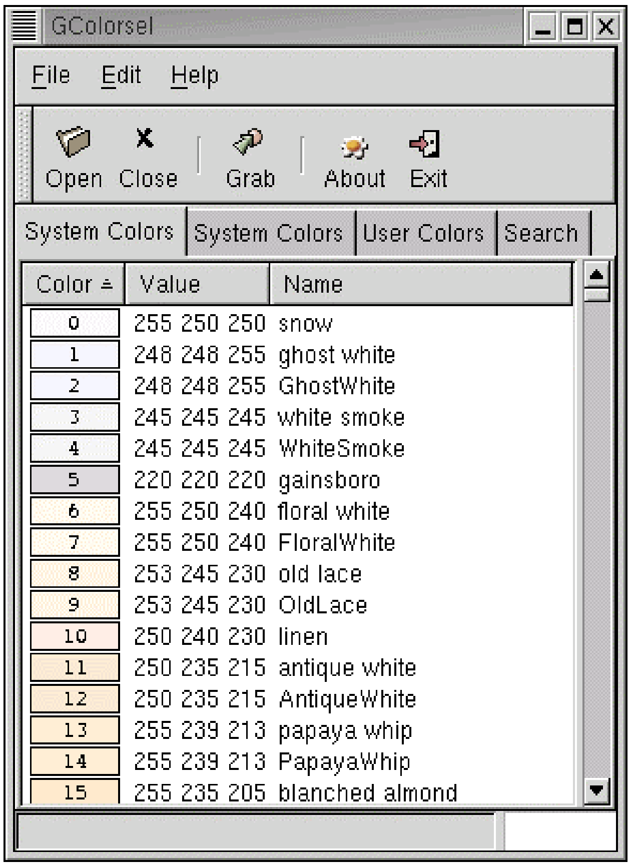
```
showrgb | grep -i orange
```

showrgb lists the colors and their decimal red, green, and blue values. The preceding command's output follows:

```
255 165   0             orange
255 140   0             dark orange
255 140   0             DarkOrange
255  69   0             orange red
255  69   0             OrangeRed
255 165   0             orange1
238 154   0             orange2
205 133   0             orange3
139  90   0             orange4
255 127   0             DarkOrange1
```

```
238 118   0            DarkOrange2
205 102   0            DarkOrange3
139  69   0            DarkOrange4
255  69   0            OrangeRed1
238  64   0            OrangeRed2
205  55   0            OrangeRed3
139  37   0            OrangeRed4
```

To actually see what the colors look like, you can use the *xcolorsel* client or *gcolorsel*, the GNOME Color Browser,[2] shown in Figure 14-1. Both display the colors defined in *rgb.txt*. and can be used to pick colors for use with a client program, either by selecting colors you like and noting their names or by finding the names of colors displayed elsewhere on your screen. This section describes *gcolorsel*; the use of *xcolorsel* is similar.



Color	Value	Name
0	255 250 250	snow
1	248 248 255	ghost white
2	248 248 255	GhostWhite
3	245 245 245	white smoke
4	245 245 245	WhiteSmoke
5	220 220 220	gainsboro
6	255 250 240	floral white
7	255 250 240	FloralWhite
8	253 245 230	old lace
9	253 245 230	OldLace
10	250 240 230	linen
11	250 235 215	antique white
12	250 235 215	AntiqueWhite
13	255 239 213	papaya whip
14	255 239 213	PapayaWhip
15	255 235 205	blanched almond

Figure 14-1: gcolorsel, the GNOME Color Browser

[2] If your version of XFree86 didn't come with *xcolorsel*, you can download it from one of the Linux download sites. Or, if GNOME is installed on your system, you can use *gcolorsel* instead (even if you aren't using GNOME as your desktop environment). *gcolorsel* is part of the *gnome-utils* package.

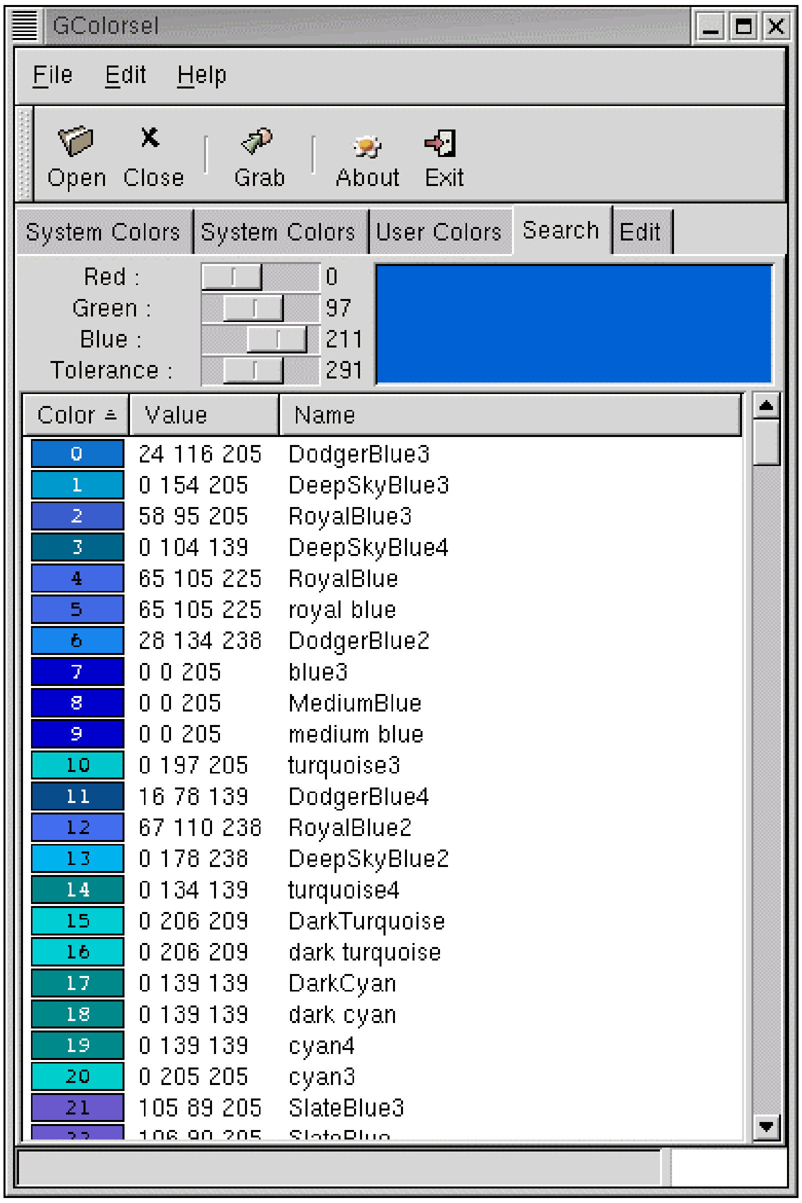
If you are running GNOME, you can run the color browser by selecting Main Menu⮙Programs⮙Utilities⮙Color Browser. Whether you are running GNOME or not, you can run *gcolorsel* from the command line in an *xterm* window:

```
$ gcolorsel &
```

As the figure shows, the window is divided into three sections. The portion that we're interested in is the main section, which fills the lower part of the window. There are four tabs at the top of this window; each tab shows a different view. (Depending on your version of *gcolorsel*, the tabs may look slightly different or be ordered differently.) You can move between the views by clicking the pointer on the appropriate tab.

Two of the tabs say System Colors. One shows the colors in *rgb.txt* as a list and the other shows them as a grid, where each color is represented as a small square of color. The figure shows the list view, displaying the beginning of *rgb.txt*. On the right is a scrollbar, which lets you scroll through the entire file. At the left on each line is a small rectangle displaying the color as it would appear on your screen. The number inside the rectangle is the line number of that color in the *rgb.txt* file. The color is followed by the decimal values for red, green, and blue for that color, and the color's name. If you are just looking for a color to use with a client, you can ignore the rest of the window and scroll through the list. When you find a color you like, note its name and you are done. If you are running GNOME, you can click on the color rectangle and drag the color to a GNOME object. For example, you can pick a color you like and drag it to the root window or the panel. The window or panel changes to that color and the new color information is saved in the GNOME configuration files, making the change permanent across login sessions.

The tab labeled Search is used to find a system color most closely matching a given color. When you click on the Search tab, the window changes to include a new section between the tabs and the list of colors. That section includes four sliders and a window with a large color-filled rectangle as shown in Figure 14-2.



Red : 0
Green : 97
Blue : 211
Tolerance : 291

Color	Value	Name
0	24 116 205	DodgerBlue3
1	0 154 205	DeepSkyBlue3
2	58 95 205	RoyalBlue3
3	0 104 139	DeepSkyBlue4
4	65 105 225	RoyalBlue
5	65 105 225	royal blue
6	28 134 238	DodgerBlue2
7	0 0 205	blue3
8	0 0 205	MediumBlue
9	0 0 205	medium blue
10	0 197 205	turquoise3
11	16 78 139	DodgerBlue4
12	67 110 238	RoyalBlue2
13	0 178 238	DeepSkyBlue2
14	0 134 139	turquoise4
15	0 206 209	DarkTurquoise
16	0 206 209	dark turquoise
17	0 139 139	DarkCyan
18	0 139 139	dark cyan
19	0 139 139	cyan4
20	0 205 205	cyan3
21	105 89 205	SlateBlue3
22	106 90 205	SlateBlue

Figure 14-2: gcolorsel's Search window

Once you've selected the Search tab, you can specify a color to match in several ways:

- Move the Red, Green, and Blue sliders to specify a set of RGB values. Note that the color of the window to the right of the RGB sliders changes automatically and the list of system colors in the lower portion of the window changes to reflect the colors closest to the one you just made. When you're satisfied with the resulting color, go to the list and find the closest match.
- Grab a color of any pixel on the screen. To use this method, click on the Grab button in the toolbar (the second row of the *gcolorsel* window). Your mouse cursor turns

into a miniature dropper. Place this dropper over the color you want on the screen and click. *gcolorsel* determines the color's RGB values, and shows the color and the RGB values in the Search window. As with the first method, it also shows the system colors closely matching this one. If you grab a color before selecting the Search tab, the tab is automatically selected for you.

- If you are running GNOME, you can drag and drop a color from some other color source, such as the panel or one of the color rectangles in the System Colors grid view. An easy way to use colors from the grid is to modify the preferences by setting Edit⮝Preferences⮝Documents⮝Mode to Toplevel. This setting replaces the tabs with individual windows, allowing you to drag-and-drop across windows. If you don't want separate windows, you can use the color rectangles in the Search tab itself.

gcolorsel shows only those system colors that closely match the one you selected, and orders them by how close they are so that the best matches are shown first. You can adjust the number of system colors displayed (the *tolerance*) by using the Tolerance slider: smaller values make *gcolorsel* more selective, so that only close matches are shown. Note that the tolerance is a number, not a percent. Setting the tolerance value to 0 shows only exact matches; setting the value to 100 shows the 100 closest colors.

The GNOME Color Browser has a number of other features that we haven't covered here, including the ability to edit colors and to create your own color set. For more information about those features, the Help menu on the menu bar (the first line of the browser window) displays the manual. If you issue the *gcolorsel* command from the command line, the *--help* option displays a usage menu showing the command-line options.

If *gcolorsel* can't find an exact match, it shows you the closest matches, and you can pick one you like. If you want an exact color match and *gcolorsel* doesn't find it, you can use *xmag* (described in more detail in Chapter 3) to find the numeric color values. To do that, use *xmag* to zero in on a part of the image. Then move the mouse to a pixel that has the color you want and press the first mouse button to display the RGB values for that pixel. The values are in hexadecimal; to use them on the command line or in a resource file, just remove the commas, preface the hexadecimal number with a pound sign (#),[*] and enclose it in double quotes (to prevent the shell from interpreting it as a special character).

```
xbiff -bg "#90ffe0" &
```

Or you can translate the hex value to the decimal RGB equivalent, give the color a name, and add it to *rgb.txt* to save it permanently. For example, our hex value of 90ffe0 above converts to an RGB value of 144 255 224 (see the section "Specifying RGB Colors as Hexadecimal Numbers" and the sidebar "The Hexadecimal Numbering System" later in

[*] Note that while you'll still see RGB values expressed in this format, it is an older, deprecated, way of expressing them. The currently approved way is to use the Xcms format, which we'll cover later. In that format, the command would be: `xbiff -bg rgb:90/ff/e0`. Also note that what *xmag* actually shows is `9090,ffff,e0e0` and that both formats are case-insensitive.

this chapter for information on how to do that.) To add this color to *rgb.txt* with the name "mint green", add the following two lines to the file with any text editor:

```
144 255 224      mint green
144 255 224      MintGreen
```

Before you can use the new color, you need to restart the X server (the server always reads *rgb.txt* when it starts). If for some reason your X server can no longer read this file, you can still use colors. However, you'll have to refer to them using hexadecimal notation instead of the descriptive names.

There's also no easy way to find the names of the default colors for your X clients. You can check your own and the system default resource files to see what colors a client is configured to use, or you can simply try different colors to see if they match. With Gnome and KDE, the configuration tools show you the current color settings, let you select colors, and let you define your own colors, but even they don't tell you the color names.

How Color Works

X client programs that draw in color use the value of the bits associated with a pixel to determine the color to be placed on the display for that pixel. The way the hardware does that is known as a *visual*. The visual provides the hardware with the information it needs to translate the pixel values into RGB colors, by using the bit values as a pointer to a lookup table internal to the X server called a *colormap*. A colormap is a fixed-length table internal to the X server. Each colormap entry, or *colorcell*, contains the red, green, and blue values for a particular color.

There are six visual types. The two that are of the most interest to XFree86 users are TrueColor and PseudoColor. All six visuals are described briefly in the following list, then TrueColor and PseudoColor are described in more detail.

DirectColor
: Each of the red, green, and blue values for the pixel independently indexes the equivalent value in the colormap. For a colormap of a given size, this allows for many more colors than PseudoColor, where the three values together index a single colorcell. The colormap is writeable.

TrueColor
: Like DirectColor, except that the colormap is read-only.

PseudoColor
: The red, green, and blue values for the pixel together index a single colorcell. The colormap is writeable.

StaticColor
: Like PseudoColor in that each pixel is a pointer into a colorcell, but the colormap is read-only.

GrayScale
: Similar to PseudoColor, but for each colorcell the red, green, and blue values are always the same, resulting in a shade of gray. The colormap entries are writeable.

StaticGray

Like GrayScale, but the colormap is read-only. StaticGray visuals are generally used for monochrome displays that have only two color choices.

The visual being used determines how XFree86 accesses the colormap; in turn, the bit depth at which X is running determines which visual is used. The best way to see which visual your system is using is to run the *xdpyinfo* command; the output from this command also tells you what other visuals are available (in addition to providing a lot of other information).

The default visual for an 8-bit display is PseudoColor; for depths greater than 8, the default is TrueColor. You can change the default by setting the `Visual` option in the `Display` subsection of the `Screen` section in the *XF86Config* file; see Chapter 10, *Configuring XFree86*, for details. Note, though, that on the PC, there is a single read-only hardware colormap that has to be shared by all applications.

PseudoColor

With PseudoColor, the colormap cells are filled as colors are entered. Every X window has an associated colormap. There is a default colormap, which many clients share. In addition, applications can define their own private colormaps.[*] Because colormaps are server-specific, if you are on a multiuser system, or you are running many X clients, each configured to use different colors, there is a possibility that you might run out of entries.

As shown in Figure 14-3, the numeric pixel values in the framebuffer[3] are used as an index into the colormap. The information in the colormap is then used to drive the hardware that actually displays the color on the screen.

Illustration department, I need help with this figure. It should look like Figure 26-2 on page 592 of X Users Tools. The following figure, Swiss Alpine Club's logo, is just a place holder. If it should get printed by mistake you would my wife a joy!

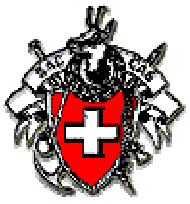


Figure 14-3: Multiple planes used to index a colormap

Why is this technical detail important? Because it explains several issues that you might encounter in working with color displays.

First, the range of colors theoretically available on the display is a function of the number of bits available in the colormap for RGB specification. If 8 bits are available to specify the value of each of the three primary colors (i.e., each color can take a value from 0 to

[*] If you see your screen flashing strange colors, it's probably because an application is switching its private colormap in or out.

[3] The framebuffer is the memory on the video card that holds a screen image to be displayed.

255), then the range of possible colors is 256^3, or more than 16 million colors. This means that you can create incredibly precise differences between colors.

However, the number of different colors that can actually be displayed on the screen at any one time depends on the number of planes (i.e., the depth). An 8-plane PseudoColor visual can only index 2^8 colorcells, or 256 distinct colors. Thus, if you are using display hardware with a color depth of 8 planes, the fact that you can precisely define colors containing 256 different values of blue is far less significant than the fact that you can't use them all at the same time. There isn't space for all of them to be stored in the colormap at one time nor does PseudoColor provide a mechanism for selecting them even if they could be stored.

This limitation is made more significant by the fact that X is a multiclient environment. When X starts up, usually no colors are initially loaded into the colormap. As clients are invoked, certain of these cells are allocated. But when all the free colorcells have been used, it is no longer possible to request new colors. When this happens, you get an error message telling you that the application cannot allocate the color. Thus, with PseudoColor, the possibility exists that an application will fail because it cannot allocate a color, but if it doesn't fail, you will always get the requested color.

To minimize the chance of running out of colorcells, many programs use *shared* colorcells. Shared colorcells can be used by any number of applications, but they can't be changed by one application as long as any other applications are still using them. They can only be deallocated by each application that uses them; when all applications have deallocated a cell, that cell is available for setting to another color. Shared cells are most often used for background, border, and cursor colors. You might also want to investigate the use of themes as a way to standardize the colors you use.

Alternatively, some clients must be able to change the color of graphics they have already drawn. This requires another kind of cell, called *private*, which can't be shared. A typical use of a private cell would be for the palette of a color-mixing application, where the primary colors don't change and therefore can use shared cells, while the color being mixed uses a private cell.

TrueColor

If you are running XFree86 at 15, 16, 24, or 32 bpp, you are most likely running with a TrueColor visual. One of the features of TrueColor is that you will always get a color, but unlike PseudoColor, it might not be the precise color that the application requested. However, this is less important with TrueColor because all the colors theoretically available are actually available all the time. In addition, because the TrueColor colormap is read-only, your application cannot change the color of a pixel once it has been written to the display. Thus you can have as many colors as you want on the display simultaneously, but your applications cannot change the color associated with a pixel.

The Color Models

Two *color models* are used with X: *RGB* and *Xcms*. RGB (Red, Green, Blue) is a server-side non-portable color model, while Xcms (the X Color Management System) is a client-side, device-independent model. RGB colors are dependent on the display hardware, and they may look different on dissimilar monitors. Xcms colors are independent of the device they are displayed on. You don't have to choose between them; you can take advantage of both models. The rest of this chapter describes these color models, how to specify colors with each model, and how to create and manage your own color database.

We've seen how to browse the standard RGB color database to find colors you like. Keep in mind, though, that there is an inherent limitation in the RGB system—the colors are created by combining different amounts of red, green, and blue. This system matches the way most monitors work, but it makes the RGB colors device-dependent. Thus a color is likely to look different when displayed on different types of monitors. Probably most users won't care if their "pink" is too "orangy" and will just experiment with other colors in the database to find shades they like.

However, if you want absolute precision of color regardless of the environment and hardware, you'll want to use the X Color Management System (Xcms) described in the section "The X Color Management System".

The RGB Color Model

Within the *rgb.txt* file, each color name is associated with three numeric values corresponding to the amounts of red, green, and blue in the shade, in that order. The display hardware uses these values to produce a color. For example, here are a few lines from *rgb.txt*:

```
255 222 173            navajo white
255 222 173            NavajoWhite
255 228 181            moccasin
255 248 220            cornsilk
255 255 240            ivory
255 255 255            white
```

You can see from the last line of the example that if all three primary colors are at their maximum value of 255, the resulting color is white. Slight variations in the values of green and blue produce shades of off-white, with names like ivory or moccasin. In addition, more than one name can be associated with a particular combination of RGB values. The duplicates are often variations of the same name, such as navajo white and NavajoWhite above, but they don't have to be.

Naming colors

Whether you're running a command or adding a resource, you have to either enter multiple-word color names like "midnight blue" as single words (e.g., midnightblue) or surround them with quotes. That's why we specified the color as `steelblue` in the *xsetroot* command earlier in this chapter; we could have issued the command as follows with the same result:

```
xsetroot -screen "steel blue" &
```

Furthermore, case doesn't matter. You can refer to steel blue, Steel Blue, SteelBlue, or steelblue and get the same color, even though *rgb.txt* contains only the lines:

```
70 130 180             steel blue
70 130 180             SteelBlue
```

You can specify an RGB color in different formats:

- A color name from the database
- The numeric color value in hexadecimal (hex) notation, prefixed with a pound sign (#)
- Using the Xcms format but specifying the RGB color space.

For example, the following commands all produce an *xbiff* window with the same foreground color:

```
xbiff -fg PeachPuff              color name
xbiff -fg rgb:ff/da/b9           hexadecimal numeric value
xbiff -fg "#ffdab9"              Xcms format
```

You can use any of the formats on the command line or in resource files—wherever you can specify a color. As we'll see later in this chapter, you can also use hex values to get colors that aren't in the database.

Advantages and disadvantages of the RGB database

From a user's standpoint, the advantages of the RGB database are that it:

- Makes available a wide range of colors whose names you can simply plug in to command lines and resource specifications.
- Allows customization but doesn't require it.

The primary disadvantage of the RGB database (and the model itself) is that the colors it defines can look very different on different types of display hardware. This is because the server accesses the database (or the numeric hex specifications) and simply applies the color values, without any tuning for the type of monitor, the platform, etc. Certain intensities of red, green, and blue might produce orange on one display and pink on another. In other words, as we mentioned earlier, RGB colors are hardware-specific and thus, non-portable.

Surveying the RGB Database

The default *rgb.txt* file contains more than 750 color-name definitions. This number is deceptive, though, since many of the color names are merely spelling or naming variants that have the same color values. Others are shades of the same color. For example, here are the entries for the variations of the color "sea green":

```
143 188 143             dark sea green
143 188 143             DarkSeaGreen
 46 139  87             sea green
 46 139  87             SeaGreen
 60 179 113             medium sea green
 60 179 113             MediumSeaGreen
 32 178 170             light sea green
 32 178 170             LightSeaGreen
193 255 193             DarkSeaGreen1
180 238 180             DarkSeaGreen2
155 205 155             DarkSeaGreen3
105 139 105             DarkSeaGreen4
 84 255 159             SeaGreen1
 78 238 148             SeaGreen2
 67 205 128             SeaGreen3
 46 139  87             SeaGreen4
```

Each of the names corresponds to a color definition consisting of the three numeric RGB values (the columns are in the order: red, green, and blue). As you can see, some of the shades are distinguished in the fairly traditional way of being called "light," "medium," and "dark." The light, medium, and dark shades of a color can probably be distinguished from one another on virtually any monitor.

Beyond this distinction, there are what might be termed sub-shades: gradations of a particular shade identified by number (SeaGreen1, SeaGreen2, etc.). Adjacent sub-shades of a color may not be clearly distinguishable on all display hardware. For example, SeaGreen1 and SeaGreen2 may look very much the same. You certainly would not choose to create a window with a SeaGreen1 background and SeaGreen2 foreground. The availability of different shades of a color lets you experiment to find one that looks good on your display.

The color names in the *rgb.txt* file are too numerous to list here. Although there are no literal dividers within the file, it can roughly be considered to fall into three sections:

- A standard spectrum of colors (red, yellow, sea green, powder blue, hot pink, etc.), that seem to be ordered roughly as: off-whites and other pale colors, grays, blues, greens, yellows, browns, oranges, pinks, reds, and purples.
- Sub-shades of the colors in the first section (such as SeaGreen 1 through 4).
- One hundred and one shades of gray, numbered 0 through 100. This large number of precisely graduated grays provides a wide variety of shading for grayscale displays.[*]

Other RGB Color Databases

If you aren't happy with the standard RGB database, the easiest solution is to modify it by adding the colors you want to *rgb.txt*. However, it is possible to substitute an alternative database. There are two ways to do this:

- Modify the line in *XF86Config* that specifies the location of *rgb.txt* and substitute the name of the new file. The standard entry (in the Files section) looks like this:

  ```
  RgbPath     "/usr/X11R6/lib/X11/rgb"
  ```

  For example, let's change that to the new path:

  ```
  RgbPath     "/home/werner/myrgb"
  ```

When we restart X, RGB colors are taken from *myrgb.txt*.

- Replace the existing *rgb.txt* with another database file. (You might want to back up the old file first, just to be safe.)

> In either of the above cases, you'll need root privileges to make the change. Also, since there is only one RGB database for the server, if you are on a multiuser system and you change the database, you'll be changing it for everyone. Adding entries is okay, but removing entries or replacing the entire database can cause problems for other users.

[*] There are actually two hundred and two entries in this section--one for each shade spelled *gray* and another for each shade spelled *grey*.

In addition to the standard RGB color database, the source distribution of X includes three alternative databases that can be used. In the XFree86 source tree, the files are in the directory *xc/programs/rgb/others*. The databases you'll find there are called *raveling.txt*, *thomas.txt*, and *old-rgb.txt*.

These databases are primarily older versions of the current standard RGB database. The color names are pretty much the same, but in some instances the numeric RGB values associated with the colors are different. So if you don't like the way a particular color appears on your monitor, you could try substituting the value in the older database to see if you like that better. But that's a lot of trouble to go to, when you can just pick a different color instead.

Specifying RGB Colors as Hexadecimal Numbers

As we've seen, each RGB color has three numeric values associated with it, that specify the amount of red, green, and blue in the color. In the *rgb.txt* file, these values are in decimal notation and are paired with a color name. For example:

```
  0   0   0            black
255 235 205            BlanchedAlmond
255 255 255            white
```

Being able to specify a color by numeric value means that you can be very precise and that you can define a virtually infinite number of shades. You are then independent of the possibly limited number of entries found in *rgb.txt*. (Note, though, that you are still constrained by the size of the colormap and the number of color cells it contains.) In addition to decimal values, the RGB model also allows you to specify colors in *hexadecimal* (commonly referred to as *hex*) notation. The additional flexibility of being able to supply the values in hex as well as decimal is handy for working with programs like *xmag* that output hex.

Note to production: The code in the sidebar is supposed to come after the paragraph that starts "For example,..." not before it. It's correct if you look at the file in Normal view, but not in Print Layout view.

The Hexadecimal Numbering System

Hex is a base-16 numbering system used in programming and in some scientific disciplines; in addition to the normal decimal characters 0-9, it also uses the letters A-F to represent the numbers 10-15. Thus, for example, the number E in hex is the same as 14 decimal, hex F is decimal 15, hex 10 is decimal 16, and hex 11 is decimal 17.

A single hex digit can be represented in four bits. The hex numbers 0 through FF are equivalent to decimal 0 through 255. These decimal 256 possibilities can be represented in eight bits, which is a byte.

Hex numbers are case-insensitive and can be specified as the capital letters A-F or the small letters, a-f. Hex AD is the same as hex ad. Often the prefix `0x` is added to indicate a hex number. Hex AD can therefore be written as 0xAD.

For example, in the section “Finding Colors”, we ran *xmag* to find the hex RGB value 90ffe0, where 90 is the hex value for red, ff is the hex value for green, and

```
90=(9x16)+0=144
ff=(15x16)+15=255
e0=(14x16)+0=224
```

e0 is the hex value for blue. Let’s translate these numbers to decimal:

The result is the RGB triplet 144 255 224. An easier way to do the conversion, especially if you want to convert from decimal to hex, is to find a calculator, such as the KDE calculator *kcalc*, that does the conversion for you. With *kcalc*, for example, you can set the base to decimal, enter a decimal value, then change the base to hex and *kcalc* converts the number and displays the hex value.

When you run *xmag* and click the left mouse button on a pixel of an interesting color, you’ll see a message like the following:

```
Pixel 1644912 at (966,84) colored (1919,1919,7070).
```

The first part of the line gives you information about the pixel you selected, which you can ignore. The rest of the line gives you the red, green, and blue color values (in this case 1919, 1919, and 7070). Then run *xbiff* using that color:

```
xbiff -fg "#191970" &
```

and you get an *xbiff* window with a mailbox (the foreground) of the same color as the one you selected in *xmag*. See the section “Inside the RGB Color Model” for an explanation of why we were able to specify only two digits for each value in the *–fg* option rather than four digits (i.e., 191970 rather than 191919197070).

Adding Colors to the RGB Database

As we’ve seen, the X Window System comes with a predefined set of colors, listed in the file */usr/X11R6/lib/X11/rgb.txt*. If you have access to a color-editing program such as *xcoloredit*, you can also come up with your own colors and add them to the standard RGB database, making them easier to refer to. This also makes them available to other users on your system. To add them, you first need to know their decimal RGB color values.

Once you have those values, you can pair them with a color name and add the color definition to *rgb.txt*. The format of a line in *rgb.txt* is:

```
red green blue   color_name
```

where `red`, `green`, and `blue` are integers in the range 0 to 255, and `color_name` is case-insensitive but cannot include any special characters or symbols. There must be a tab (not spaces) separating the values from the name, but the name can include spaces.

Say you want to define a color called tropical blue, with RGB values of 9, 229, and 251. Since the name is composed of more than one word, you need to make two entries: one as multiple words (tropical blue) and one as a single word (TropicalBlue).

To update the RGB database, you just edit *rgb.txt* to add the new color specification. The new lines can go anywhere in the file and will look like this:

```
9 229 251       tropical blue
9 229 251       TropicalBlue
```

If you add a color with a one-word name, a single line will do:

```
9 229 251       caribbean
```

The next time you restart X, any new colors you added will be available.

Inside the RGB Color Model

Most color displays on the market today, whether CRTs or LCDs, are based on the RGB color model.

On a CRT, each pixel on the screen is actually made up of three phosphors: one red, one green, and one blue. Each of these three phosphors is illuminated by a separate electron beam, called a *color gun*. These color guns can be lit to different intensities to produce different colors on the screen.

LCD displays use three liquid-crystal cells to provide the color for a single pixel. Each cell has a red, green, or blue filter over it. Electrical impulses light the cells, transmitting the filtered color to that pixel.

In either case, numeric red, green, and blue values determine the amount of light that passes through each phosphor or liquid crystal cell, which in turn determines the intensity of one of the primary colors. The relative amounts of each of the three primary colors determine the final color for one pixel. In the *rgb.txt* file, each color is associated with a decimal number between 0 and 255. Consider the following line from the *rgb.txt* file:

```
173 216 230              light blue
```

The three numbers make up what is known as an *RGB triplet*. As we've seen, the *rgb.txt* file contains more than 750 mappings of RGB triplets to color names.

When all three colors are fully illuminated (the triplet values are 255, 255, 255), the pixel appears white to the human eye. When all three are dark (the triplet values are 0, 0, 0), the pixel appears black. As the illumination of each primary color varies, other colors are produced. For example, equal portions of red and green, with no admixture of blue, makes yellow.

An RGB triplet can also be supplied in hexadecimal notation, which permits greater precision than the decimal values in *rgb.txt*. Depending on the underlying hardware, different servers may use a larger or smaller number of bits (from 4 to 16) to describe the intensity of each primary. To insulate you from this variation, most clients are designed to take color values containing anywhere from 4 to 16 bits (1 to 4 hex digits), and the server then scales them to the hardware. As a result, you can specify hexadecimal values in any one of these formats:

```
#RGB
#RRGGBB
#RRRGGGBBB
#RRRRGGGGBBBB
```

Or in the Xcms format:[4]

```
rgb: R/G/B
rgb: RR/GG/BB
rgb: RRR/GGG/BBB
rgb: RRRR/GGGG/BBBB
```

where R, G, and B represent single hexadecimal digits and determine the intensity of the red, green, and blue primaries that make up each color. As you add more bits, you get greater precision and finer gradations of color.

When fewer than four digits are used, they represent the most significant bits of the value. For example, `#3a6` is the same as `#3000a0006000`.

What this means concretely is perhaps best illustrated by looking at the hex values that correspond to some colors in the color-name database. We'll use 8-bit values—two hexadecimal digits for each primary. These definitions are the hexadecimal equivalents of the decimal values for some of the colors found in the *rgb.txt* file:

```
#000000     black
#FFFFFF     white
#FF0000     red
#00FF00     green
#0000FF     blue
#FFFF00     yellow
#00FFFF     cyan
#FF00FF     magenta
#5F9EA0     cadet blue
#6495ED     cornflower blue
#ADD8E6     light blue
#B0C4DE     light steel blue
#0000CD     medium blue
#000080     navy blue
#87CEED     sky blue
#6A5ACE     slate blue
#4682B4     steel blue
```

As you can see, pure red, green, and blue result from the corresponding bits being turned on fully. Turning all the primaries off yields black, while turning all on produces white. Yellow, cyan, and magenta can be created by pairing two of the other primaries at full intensity. The various shades of blue are created by varying the intensity of each primary.

Of course, fiddling with the numbers is fairly unintuitive. If you want to play with color, use a color editor.

[4] Xcms is described in the next section, "The X Color Management System".

Xcms, the X Color Management System

If you're not entirely satisfied with the hardware-specific RGB model, an additional, more precise color model is available. The X Color Management System (Xcms), developed by Tektronix and later adopted by the X Consortium, was designed to overcome the limitations of the RGB model by providing *device-independent* color. Under Xcms, color definitions are based upon internationally recognized standards. The idea behind Xcms is that color relies upon human vision. Simply put, red should look basically the same on any monitor, on any platform.

Xcms accepts color values in several different formats, called *color spaces*. Most of these color spaces describe color in a device-independent manner, using scientific terms and values commonly applied to color. Among the color spaces that Xcms recognizes are two RGB color spaces; however, the RGB colors are still not portable even when specified as Xcms color spaces. As with the RGB model, you can specify the Xcms color spaces on the command line and in resource files. An example of a color in the RGB color space specified on the command line as an Xcms color is:

```
xbiff -fg rgb:19/19/70
```

From a user's standpoint, the primary advantages of Xcms are that it:

- Recognizes several types of color specification (color spaces, described in the next section, "The Xcms Color Spaces), which can be supplied on the command line or in resource files.
- Enables you to make a database of colors you "mix" yourself, using a color editor. (While the server-side RGB model allows for a single system-wide database, Xcms allows any user to have a private database.) In the Xcms database, you pair a name with a value in any of the accepted color spaces (formats). The Xcms database can then serve as an alternative to the default RGB database (you can specify colors from either).
- Provides the ability to "tune" colors to display more accurately on specific hardware.
- Allows you to take advantage of sophisticated color printer technology.

Though your Linux system certainly supports the Xcms color management system and it has its definite advantages, most users prefer the RGB color model. It's familiar and for the average user, the greater precision of Xcms is not necessary. If you are simply looking for colors for your client windows, use the RGB colors. However, if you are developing an application, or you do serious graphics work, you may want to use Xcms to be sure of getting the colors you intend.

The Xcms Color Spaces

Under Xcms, each color specification has a prefix that indicates the color space and a numeric value that specifies the color. Table 14-1 summarizes the color spaces and their prefixes.

Table 14-1: Xcms Color Spaces

Name	**Prefix**

Tektronix HVC*	TekHVC
Various CIE* formats	CIEXYZ, CIEuvY, CIExyY, CIELab, CIELuv
RGB	RGB
RGB Intensity[5]	RGBi

Of the valid color spaces, the Tektronix HVC and the various CIE formats specify color in a device-independent manner, while the RGB color spaces are hardware-specific, as we've seen. Therefore, to take advantage of the portability of the X Color Management System, you'll want to use TekHVC or one of the CIE formats.

To specify a color under Xcms, you combine its numeric value with the appropriate prefix for the color space you are using. The syntax of an entry in the Xcms database is:

```
prefix:value1/value2/value3
```

The following are examples of valid Xcms color specifications:

```
CIEuvY:0.15259/0.40507/0.44336
TekHVC:223.93036/72.45283/29.67013
RGB:6a/bb/d8
```

These three sample values all define a shade of blue, each using a different notation. We can supply *any* of these color specifications on a given display and get the same color. Thus, the following command lines should produce identical *xbiff* windows:

```
xbiff -fg CIEuvY:0.15259/0.40507/0.44336 &
xbiff -fg TekHVC:223.93036/72.45283/29.67013 &
xbiff -fg RGB:6a/bb/d8 &
```

Note that like the RGB color names, color specifications in an Xcms color space format are case-insensitive. Thus, `rgb:6a/bb/d8` and `RGB:6A/BB/D8` are equivalent.

Because the RGB colors aren't portable even when specified in Xcms format, if we want to display this precise shade of blue on another monitor we have to use one of the portable specifications:

```
CIEuvY:0.15259/0.40507/0.44336
TekHVC:223.93036/72.45283/29.67013
```

You can also use any valid color space as the value of a resource variable:

```
xbiff*foreground: TekHVC:223.93036/72.45283/29.67013
```

It's handy to be able to plug these numbers into a command line or resource specification, but if you want to use your own colors on a regular basis, it's a good idea to create an Xcms database—then you can refer to the colors by name.

* Tektronix developed the X Color Management System; the initials HVC refer to hue, value, and chroma, the scientific characteristics of color.

* CIE stands for *Commission Internationale de l'Eclairage* (or *International Commission on Illumination*), an international standards organization.

[5] RGB Intensity expresses the intensity of the red, green, and blue values linearly as floating-point values from 0.0 to 1.0 (with an optional sign and exponent), where 1.0 means full intensity of the color.

Creating an Xcms Color Database

This section tells you how to create your own Xcms color database. You should note, though, that there is one limitation: you cannot access an Xcms database across the network. In other words, if you create an Xcms color database on your own workstation, you can't use any of those color definitions in a remote process. This limitation has nothing to do with whether a particular color specification is portable. When we say that an Xcms color is portable, we mean that it will look the same on any system. However, you must specify the color on that system or have access to the color definition in a database local to the client in order to use that color.

Creating an Xcms color database is easy. A sample Xcms database is provided in the XFree86 source tree as *xc/lib/X11/Xcms.txt*. The format of an Xcms color database is:

```
XCMS_COLORDB_START 0.1
color_name<tab>color_space
                .
                .
                .
XCMS_COLORDB_END
```

The first and last lines are entered exactly as shown here. The `0.1` at the end of the first line indicates the version of the file's format and is required. Between the first and last lines you can put any number of color definitions. A text name for the color goes in the first column, followed by a tab (this is also important), and then a valid color space. Note that the arrangement of the columns is the opposite of that in the *rgb.txt* file, in which the numeric values come first and the text name second.

By default, Xcms looks for */usr/X11R6/lib/X11/Xcms.txt*. You can specify an alternative database file by setting the XCMSDB environment variable. This enables every user to have a private color database. Note, however, that Xcms only checks one database. If you set XCMSDB to another file, Xcms does not check */usr/X11R6/lib/X11/Xcms.txt*, even if it exists. (In other words, you cannot specify your own private colors and also take advantage of system-wide Xcms definitions.)

If you're on a multiuser system, you might choose to create an Xcms database for all users to access, putting it into the directory */usr/X11R6/lib/X11* (where *rgb.txt* also generally can be found). In this case, the database file should be named *Xcms.txt*.

If you are creating a personal Xcms database, you can give it any name you want; then you just need to set the XCMSDB environment variable to the full pathname of the file. For example, say you create some colors you'd like to use in your normal X session (for the root window, window titlebars, etc.). You might put these into an Xcms file in your home directory called *.xcolors*. Then specify one of the following, depending on your shell:

```
export XCMSDB=~/.xcolors     bash
setenv XCMSDB ~/.xcolors     tcsh
```

so the X server can find it. You probably want to put this in your session file, or in your *.bashrc* or *.tcshrc* file, so it runs every time you log in.

Here are the color entries from a sample database file:

```
XCMS_COLORDB_START 0.1
red                     CIEXYZ:0.3811/0.2073/0.0213
```

```
green                   CIEXYZ:0.3203/0.6805/0.1430
blue                    CIEXYZ:0.2483/0.1122/1.2417
aquamarine              CIEXYZ:0.5512/0.7909/0.9759
cadet blue              CIEXYZ:0.2218/0.2815/0.4708
cornflower blue         CIEXYZ:0.3400/0.3109/1.1067
navy blue               CIEXYZ:0.0478/0.0216/0.2392
navy                    CIEXYZ:0.0478/0.0216/0.2392
brown                   CIEXYZ:0.1333/0.0772/0.0217
gray0                   TekHVC:0.0/0.0/0.0
gray10                  TekHVC:0.0/10.0/0.0
gray20                  TekHVC:0.0/20.0/0.0
gray30                  TekHVC:0.0/30.0/0.0
gray40                  TekHVC:0.0/40.0/0.0
gray50                  TekHVC:0.0/50.0/0.0
gray60                  TekHVC:0.0/60.0/0.0
gray70                  TekHVC:0.0/70.0/0.0
gray80                  TekHVC:0.0/80.0/0.0
gray90                  TekHVC:0.0/90.0/0.0
gray100                 TekHVC:0.0/100.0/0.0
XCMS_COLORDB_END
```

Like the RGB names, Xcms color names are case-insensitive. The sample database includes only portable CIE and TekHVC color specifications, but you can also include RGB color spaces in your *Xcms.txt* file.

Once *Xcms.txt* has been set up, you can specify any of the color names it contains. Thus, you can immediately enter the command line:

```
xsetroot -solid aquamarine &
```

Multiple-word color names must be specified as a single word or be surrounded by quotes.

Inside the Xcms Model

While Xcms technology is complex, and a complete explanation is beyond the scope of this book, we'll mention some of its features that we haven't already covered here. There are actually two components to Xcms:

- The color spaces or formats (that should look the same on any system)
- Optional *Device Color Characterization* (DCC) data that "tunes" color specifications for a particular hardware display.

We've already taken a look at some of the valid color spaces. If you specify a color using one of the portable Xcms color spaces, you should get the same color regardless of the monitor, server, etc.

The color spaces alone should be sufficient for most users. However, if you are developing applications, you might choose to install a DCC file (also called a *Device Profile*), to fine-tune Xcms colors for a particular hardware display. This tuning is an optional part of the system. You would probably need two adjacent monitors to see the difference between a system accessing DCC data and one not..

The DCC data is stored in properties on the screen's root window. Some servers can automatically load the properties with data appropriate to the attached display(s), but you may have to run the *xcmsdb* client to load the DCC data from a text file you specify.

Two sample DCC files come in the XFree86 source tree; you'll find them in the directory *xc/programs/xcmsdb/datafiles* with the very original names *sample1.dcc* and *sample2.dcc*.

The top portion of a DCC file (following some comments) gives a description of the monitor. For example, the following lines appear in *sample2.dcc*:

```
SCREENDATA_BEGIN        0.3

    NAME                Tektronix 19" (Panasonic) CRT
    PART_NUMBER         119-3916-00
    MODEL               XP27
    SCREEN_CLASS        VIDEO_RGB
    REVISION            2.0
                  ...
```

The remainder of the file provides data about the monitor's color capabilities. This data is loaded into the root window properties and then plugged into Xcms functions, allowing each device-independent color value to be converted into a device-specific value. You load a DCC file using the program *xcmsdb*, in the directory */usr/X11R6/bin*. Using the sample file above, the command is:

```
xcmsdb sample2.dcc
```

You would typically load the DCC file in your X startup script.

Using Both RGB and Xcms

Even if you use both RGB and Xcms color models, in practice you shouldn't have to think too much about it. You can supply color specifications in any form acceptable to either color model, and X resolves any possible conflicts.

X searches to match a color specification in this order:

1. If the specification begins with the pound sign (#), the subsequent number is interpreted as a hexadecimal RGB value.
2. If the specification contains a colon (`:`), the prefix is checked to see if it matches a valid Xcms color space; if it does, the subsequent number is interpreted as a value in that color space. All currently valid color spaces recognize the forward slash (`/`) as the delimiter between numeric values. (Each color space defines its own delimiter, so hypothetically new formats may recognize other delimiters.)
3. If the specification contains neither a pound sign nor a colon, it is assumed to be a color name that should appear in either an Xcms database or the RGB server database. The Xcms database is checked first if it exists; thus if a color name appears in both databases, the Xcms color value will be used.