CHAPTER 11

# Command Line Options

This chapter describes command line options that are common to most clients. Some arguments to command line options can also be specified as the values of resource variables, described in Chapter 12, *Setting Resources*. For example, the format of a geometry string or a color specification is the same whether it is specified as an argument to an option or as the value of a resource definition.

As explained in Chapter 3, *Opening Additional Windows*, The X Window System allows the user to specify numerous (very numerous!) command line options when starting most clients. In order to know for sure what options a given client will accept, you need to check its reference page in Part Three. However, there are many “generic” options. In addition to certain client-specific options, all applications built with the MIT X Toolkit (or a toolkit based on the Xt Intrinsics, such as OLIT, the OPEN LOOK Intrinsics Toolkit, or OSF/Motif) accept certain standard options, which are listed in Table 11-1. Programs built using the XView toolkit accept most of these options, as well as certain XView-specific options and some “-W” short forms shown in Table 11-2, “XView Generic Options”. Some non-Toolkit applications may also recognize some or all of these options. The first column contains the name of the option, the second the name of the resource to which it corresponds (see Chapter 12, *Setting Resources*), and the third a brief description of what the option does.

This chapter discusses some of the more commonly used generic options and demonstrates how to use them. (For the syntax of the other X Toolkit options, see the *X* reference page in Part Three of this guide; for the XView options, see the *xview* reference page.)

**Table 11-1. MIT Toolkit and OLIT Standard Options**

| Flag | Long Flag | Resource | Description |
|---|---|---|---|
| `-bg` | `-background` | background | Background color of window. |
| `-bd` | `-bordercolor` | borderColor | Color of window border. |

**Table 11-1. MIT Toolkit and OLIT Standard Options**

| Flag | Long Flag | Resource | Description |
|---|---|---|---|
| `-bw` | `-borderwidth` | borderWidth | Border width of window in pixels. |
| | `-display` | display | Display on which to run client. |
| `-fn` | `-font` | font | Font for text display. |
| `-fg` | `-foreground` | foreground | Foreground (drawing or text) color of window. |
| | `-geometry` | geometry | Geometry string for window size and placement. |
| | `-iconic` | | Start the application in iconified form. |
| | `-name` | name | Specify a name for the application being run. |
| `-rv` | `-reverse` | reverseVideo | Reverse foreground and background colors. |
| `+rv` | | reverseVideo | Don't reverse foreground and background. |
| | `- selection-Timeout` | selectionTim-eout | Time-out in milliseconds within which two communicating applications must respond to one another for a selection request. |
| | `-synchronous` | synchronous | Enable synchronous debug mode. |
| | `+synchronous` | synchronous | Disable synchronous debug mode. |
| | `-title` | title | Specify a window title (e.g., to be displayed in a titlebar). |
| | `-xnllanguage` | xnlLanguage | The language, territory, and code set for National Language Support; this information helps resolve resource and other filenames. |
| `-xrm` | | value of next argument | Next argument is a quoted string containing a resource manager specification, as described in Chapter 12, *Setting Resources*. |

Though all Toolkit options are preceded by a minus sign, client-specific options may or may not require one. See the reference page for each client in Part Three of this guide for the syntax of all options.

Here are the XView options. Most are preceded by a minus sign, but a few may be preceded by a plus sign to have the opposite effect.

**Table 11-2. XView Generic Options**

| Flag | Long Flag | Argument Type | Description |
|---|---|---|---|
| `-WH` | `-help` | | Print this table. |
| `-Ww` | `-width` | columns | Window width in pixels |
| `-Wh` | `-height` | lines | Window height in pixels |
| `-Ws` | `-size` | x y | (combination of above) |
| `-Wp` | `-position` | x y | Position of top left window corner |
| | `-geometry` | X geometry | X-style size, location |
| `-WP` | `-icon_position` | x y | Icon location |
| `-Wl` | `-label` | string | Window Label |
| | `-title` | string | Same as -label |
| | `-name` | string | Set application instance name to string |
| `-Wi` | `-iconic` | Start up iconified | |
| `+Wi` | `+iconic` | Start up open | |
| `-Wt` | `-font or -fn` | font name | Normal font name |
| `-Wx` | `-scale` | small \| medium \| large \| extra_large | Relative size |
| `-Wf` | `-foreground_color` | red green blue | 0-255, 0=no color; 255=full color |
| `-fg` | `-foreground` | color name | X Color specification |
| `-Wb` | `-background_color` | red green blue | same as -Wf |
| `-bg` | `-background` | color name | same as -fg |
| `-rv` | `-reverse` | | Reverse -fg and -bg colors |
| `+rv` | `+reverse` | | Don't reverse colors |
| `-WI` | `-icon_image` | filename | Take icon image from file |

Table 11-2. XView Generic Options

| Flag | Long Flag | Argument Type | Description |
|---|---|---|---|
| -WL | -icon_label | string | Label when iconified |
| -WT | -icon_font | font name | Name of font for -WL |
| -Wr | -display | server_name:screen | X display name |
| | -visual | StaticGray \| Gray-Scale \| StaticColor \| PseudoColor\| TrueColor \| DirectColor | "Visual" to use |
| -depth | depth | 1-24 | Depth of visual to use |
| -Wdr | -disable_retained | | Turn off "retained" hint |
| -Wdxio | -disable_xio_error_handler | | Turn off XLib I/O handling |
| -Wfsdb | -fullscreen-debug | | For debugging; avoid screengrabs |
| -Wfsdbs | -fullscreende-bugserver | | Avoid server grabs only |
| -Wfsdbp | -fullscreende-bugptr | | Avoid pointer (mouse) grabs only |
| -Wfsdbk | -fullscreende-bugkbd | | Avoid keyboard grabs only |
| -WS | -defeateventse-curity | | Turn off XLib event security (don't!) |
| -sync | -synchronous | | Force a synchronous XLib connection |
| +sync | +synchronous | | Make an asynchronous XLib connection |
| -Wd | -default | resource value | Set the X resource to value |
| | -xrm | resource:value | Set the X resource to value |
| | -lc_basiclocale | locale | Set basic locale to locale |

**Table 11-2. XView Generic Options**

| Flag | Long Flag | Argument Type | Description |
|---|---|---|---|
| | `-lc_displaylang` | locale | Set display language to locale |
| | `-lc_inputlang` | locale | Set input language to locale |
| | `-lc_numeric` | locale | Set numeric format to locale |
| | `-lc_timeformat` | locale | Set time format to locale |

One very useful option is `-help`, which prints the list of options that a given program will accept, similar to the above table.†

Perhaps the next-most useful options are `-display` and `-geometry`, which allow you to specify the display on which a client window should appear, and the size and position of that window, respectively. See Chapter 3, *Opening Additional Windows*, for detailed instructions on using these options. In the remainder of this chapter we'll discuss some of the other useful Toolkit options.

An option that lets you specify any X Resource for which there is no command line argument is `-xrm`, which takes an argument of the form *resource:value* (XView clients also accept -Wd or -default, which takes two arguments, a resource and a value). The two following forms are equivalent:

```
cmdtool -xrm "OpenWindows.ScrollbarPlacement:right"
cmdtool -Wd "OpenWindows.ScrollbarPlacement:right"
```

In either case, the quotes are recommended to prevent the shell from trying to interpret any asterisk ("*") or other special characters.

## 11.1 Window Title and Application Name

The name of the program (as known to the server) and the title of the window can be specified on the command line. The `-title` option allows you to specify a text string as the title of the application's window. If your application has a titlebar or if the window manager you are using puts titlebars on windows, this string will appear in the titlebar. Window titles can be useful in distinguishing multiple instances of the same application.

The `-name` option actually changes the name by which the server identifies the program. Changing the name of the application itself (with the `-name` option) affects the way the application interprets resource files. This option is discussed further in Chapter 12, *Setting Resources*. If a name string is defined for an application, that string will appear as the application name in its icon.

† Indeed, my one complaint about the standard OpenWindows clients is that they only print the generic help; many of those that accept additional command line arguments don't display them when **-help** is given.

If you display information about currently running windows using the *xwininfo* or *xlswins* client, title strings will appear in parentheses after the associated window ID numbers. (If there is no title string but there is a name string, the name string will be displayed.)

You can also use the *xwininfo* client to request information about a particular window by title, or name, if no title string is defined, using that application's own `-name` option. See the *xlswins* and *xwininfo* reference pages in Part Three of this guide and the section "Window and Display Information Clients" in Chapter 8, *Other Standard Clients*, to learn more about these clients.

# 11.2 Starting a Client Window as an Icon

The `-iconic` command line option starts the client window in iconified form. To start an *xterm* window as an icon, type:

```
% xterm -iconic &
```

XView clients accept the alternate form `-Wi, for example,`

```
% cmdtool -Wi &
```

This can be especially useful for starting a login terminal emulator window. As described in Chapter 3, *Opening Additional Windows*, terminating the login *xterm* window may kill the X server and all other clients that are running. It's always possible to terminate a window inadvertently by selecting the wrong menu option or typing the wrong key sequence. If your login *xterm* window is automatically iconified at start-up, you are far less likely to terminate the window inadvertently and end your X session.

All toolkits let you specify the location of the main window on the command line, for example using the `-geometry` command line argument. XView clients also allow you to specify the size and position of the icon on the command line. Unfortunately, the X Toolkit does not allow this. For most such clients, the size and position of the icon can be set using resource variables in an *.Xdefaults* or other resource file. (Setting the icon geometry in a resource file is highly recommended if you are starting the login *xterm* window as an icon.) See the appropriate client reference pages in Part Three of this guide for a complete list of available resources. Refer to Chapter 12, *Setting Resources*, for instructions on how to set resources.

# 11.3 Specifying Fonts on the Command Line

Many clients allow you to specify the font to be used when displaying text in the window. (These are known as *screen fonts* and are not to be confused with *printer fonts*.) For clients written with either toolkit (OLIT or XView), the option to set the display font is `-fn`. For example, the command line:

```
% cmdtool -fn fixed &
```

creates an *xterm* window in which text will be displayed with the font named ***fixed***.

Chapter 10, *X11, OPEN LOOK and OpenWindows Font Specification*, describes the available screen fonts and font naming conventions. See also *xlsfonts*.

# 11.4 Reverse Video

There are three options to control whether the application will display in reverse video—that is, with the foreground and background colors reversed. The `-rv` or `-reverse` option is used to request reverse video.

The `+rv` option is used to override any reverse video request that might be specified in a resource file. (See Chapter 12, *Setting Resources*.) This is important, because not all clients handle reverse video correctly, and even those that do usually do so only on black and white displays.

# 11.5 Specifying Color

Many clients have options that allow you to specify the color of the window background, foreground (the color in which text or graphic elements will be displayed), and border. These options generally have the form:

`-bg` ***color*** Sets the background color.

`-fg` ***color*** Sets the foreground color.

`-bd` ***color*** Sets the border color.

By default, the background of an application window is usually white and the foreground black, even on color workstations. You can specify a new color using either the color names listed in a system file called *rgb.txt* (described later) or hexadecimal values representing colors. The hexadecimal format is described in the section *Hexadecimal Color Specification* later in this chapter.

Many X Toolkit clients accept a `-bd` option that is intended to specify the color of the window border. However, under OPEN LOOK using the OPEN LOOK Window Manager, this customization is generally useless: the *olwm* frame effectively replaces most window borders. As an alternative, you can change the color of all windows' frames by specifying resources for *olwm* in a *.Xdefaults* or *.Xresources* file in your home directory. For more information, see Chapter 13, *Customizing olwm*, and the *olwm* reference page in Part Three of this guide.

In the next section, we'll take a look at some of the color names you can use. For now, let's consider the syntax of a command line specifying an *xterm* to be displayed in three colors:

```
% xterm -bg lightblue -fg darkslategrey -bd plum &
```

This command creates an *xterm* window with a background of light blue, foreground of dark slate gray, and window border of plum.

At the command line, a color name should be typed as a single word (for example, `darkslategray`). However, you can type the words that make up a color name separately if you enclose them in quotes, as in the command line:

```
% xterm -bg "light blue" -fg "dark slate grey" -bd plum &
```

As we'll see, the *rgb.txt* file contains variants of the same color name (for example, "navy blue" and "navyblue," or "grey" and "gray") to allow a range of spelling, spacing, and capitalization on the command line.

Some clients allow additional options to specify color for other elements, such as the cursor, highlighting, and so on. See the appropriate client reference pages in Part Three of this guide for details.

To see how a given color looks "in the large", you can put it onto your root window using the *xsetroot* client described in Chapter 14, *Customization Clients*. For example,

```
% xsetroot -solid "light sea green"
```

### 11.5.1 The rgb.txt File

The *rgb.txt* file, usually located in */usr/lib/X11*, is supplied with the standard distribution of X and consists of predefined colors assigned to specific text names.

A corresponding compiled database lives in two files called *rgb.dir* and *rgb.pag*. This database contains the definitions used by the server; this machine-readable file serves as a color name database and is discussed more fully in  The *rgb.txt* file is the human-readable equivalent.

### 11.5.2 X11 Release 4 Color Names

The default *rgb.txt* file shipped with Release 4 of X contains 738 color name definitions. This number is slightly deceptive, since a number of the color names are merely variants of another color name (differing only in spelling, spacing, and capitalization).

Still, the number of colors available in Release 4 is more than double the number available in Release 3. Some of the Release 4 colors are entirely new (such as snow and misty rose) but many are just slightly different shades of colors available in prior releases.

For example, the Release 3 *rgb.txt* file includes the color sea green. The Release 4 *rgb.txt* file offers several shades of that color:

```
light sea green
sea green medium
sea green dark
sea green
SeaGreen1
SeaGreen2
SeaGreen3
SeaGreen4
DarkSeaGreen1
DarkSeaGreen2
DarkSeaGreen3
DarkSeaGreen4
```

Each of these names corresponds to a color definition. (This list does not include the variants SeaGreen, LightSeaGreen, MediumSeaGreen, and DarkSeaGreen, which also appear in the file.) As you can see, some of these 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.). Numerically adjacent sub-shades of a color may not be clearly distinguishable on all monitors. For example, SeaGreen1 and 2 may look very much the same. (You certainly would not choose to create a window with a SeaGreen1 background and SeaGreen2 foreground! On the other hand, sub-shades a couple of numbers apart are probably sufficiently different to be used on the same window.)

By supplying many different shades of a single, already fairly precise color like sea green, X developers have tried to provide definitions that work well on a variety of commonly used monitors.†

You may have to experiment to determine which colors (or shades) display best on your monitor.

The color names in the Release 4 *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:

**Section 1:** A standard spectrum of colors, many available in or similar to colors in Release 3 (such as sea green). These colors seem to be ordered roughly as: off-whites and other pale colors, greys, blues, greens, yellows, browns, oranges, pinks, reds, and purples.

**Section 2:** Sub-shades of Section 1 colors (such as SeaGreen 1 through 4). These sub-shades make up the largest part of the file.

**Section 3:** One hundred and one additional shades of grey, numbered 0 through 100 (also available in Release 3). This large number of precisely graduated grays provides a wide variety of shading for monochrome screens.

Rather than list every color in the *rgb.txt* file, we've compiled this table of representative colors. We've chosen some of the more esoteric color names. Naturally all of the primary and secondary colors are also available.

```
Section 1:
ghost white peach puff lavender blush lemon chiffon slate grey midnight
blue cornflower blue medium slate blue dodger blue powder blue turquoise
pale green lawn green chartreuse olive drab lime green khaki light
yellow goldenrod indian red sienna sandy brown salmon coral tomato hot
pink maroon violet red magenta medium orchid blue violet purple
Section 2:
snow1 - 4 bisque1 - 4 cornsilk1 - 4 honeydew1 -4 azure1 - 4 SteelBlue1
- 4 DeepSkyBlue1 - 4 LightCyan1 - 4 PaleTurquoise1 - 4 aquamarine1 - 4
PaleGreen1 - 4 DarkOliveGreen1 - 4 SpringGreen1 -4 gold1 - 4 RosyBrown1
- 4 burlywood1 - 4 chocolate1 - 4 firebrick1 - 4 DarkOrange1 - 4
OrangeRed1 - 4 DeepPink1 - 4 PaleVioletRed1 - 4 plum1 - 4 DarkOrchid1 - 4
```

† The color database shipped with prior releases of X was originally designed to display optimally on the vt240 series terminals manufactured by Digital Equipment Corporation.

```
Section 3:
grey0 (gray0)  through grey100 (gray100)
```

If you want to look more closely at the *rgb.txt* file, you can open it with any text editor. As an alternative, you can display the contents of the file using the *showrgb* client. *showrgb* seems to do nothing more than *cat* (1) the file to your terminal window; in fact, it consults the database (dbm) version of the file. Given the size of the file, you should pipe the command's output to a paging program, such as *pg* (1) or *more* (1), as shown below:

```
% showrgb | more
```

See Volume Eight, *X Window System Administrator's Guide*, for information on customizing color name definitions.

Keep in mind that colors look different on different monitors. The *xcol* and *xcoloredit* clients, from the user-contributed distribution, allow you to display the colors defined in the *rgb.txt* file. *xcol* can also edit the color specifications in a resource file. See the *xcol* client reference page in Part Three of this guide.

### 11.5.3 Alternative MIT X11 Release 4 and 5 Color Databases

In addition to the standard color database described previously, Release 4 also includes three other databases your system administrator can compile. These files can be found in the general release in the directory *./rgb/others*.

*raveling.txt* Designed by Paul Raveling, this database rivals the default database in size and scope but was tuned to display optimally on Hewlett-Packard monitors.

*thomas.txt* Based on the Release 3 database, this file has been modified by John Thomas of Tektronix to approximate the colors in a box of Crayola Crayons.

*old-rgb.txt* This is nothing more than the Release 3 database.

### 11.5.4 MIT X11R5 Color Extensions

The MIT X11 Release 5 system contains numerous extensions for color editing and specification. These are not described here yet.. If your system supports them, you will find a full discussion in Volume Eight, *X Window System Administrator's Guide*.

### 11.5.5 Hexadecimal Color Specification

You can specify colors more exactly than is possible with the names in the *rgb.txt* file by using a hexadecimal color string. You probably won't use this method unless you require a color not available by using a color name. In order to understand how this works, you may need a little background on how color is implemented on most workstations.

#### 11.5.5.1 The RGB Color Model

Most color displays on the market today are based on the RGB color model. 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 excited by a separate electron beam. When all three phosphors are fully illuminated, the pixel appears white to the human eye. When all three are dark, the pixel appears black. When the illumination of each primary color varies, the three phosphors generate a subtractive color. For example, equal portions of red and green, with no admixture of blue, makes yellow.

As you might guess, the intensity of each primary color is controlled by a three-part digital value—and it is the exact makeup of this value that the hexadecimal specification allows you to set.

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
```

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.

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 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 from the colors previously given, pure red, green, and blue result from the corresponding bits being turned on fully. All primaries off yields black, while all nearly full on gives white. Yellow, cyan, and magenta can be created by pairing two of the other primaries at full intensity. The various shades of blue shown previously are created by varying the intensity of each primary—sometimes in unexpected ways.

† If you are unfamiliar with hexadecimal numbering, see the Glossary for a brief explanation, or a basic computer textbook for a more extended discussion.

The bottom line here is that if you don't intimately know the physics of color, the best you can do is look up existing colors from the color name database and experiment with them by varying one or more of the primaries till you find a color you like. Unless you need precise colors, you are probably better off using color names.

### 11.5.5.2 How Many Colors are Available?

The number of distinct colors available on the screen at any one time depends on the amount of memory available for color specification. (The *xdpyinfo* client provides information about a display, including the number of colors available at one time. See Chapter 8, *Other Standard Clients*, and the *xdpyinfo* reference page in Part Three of this guide for details. As well, the Sun OpenWindows server lets you control the number of colors with its **-cubesize** command line argument; see the *OpenWindows Server Guide* for details.)

A color display uses multiple bits per pixel (also referred to as multiple planes or the *depth* of the display) to select colors. Programs that draw in color use the value of these bits as a pointer to a lookup table called a *colormap*, in which each entry (or *colorcell*) contains the RGB values for a particular color.* As shown in Figure 11-1, any given pixel value is used as an index into this table—for example, a pixel value of 16 will select the 16th colorcell.

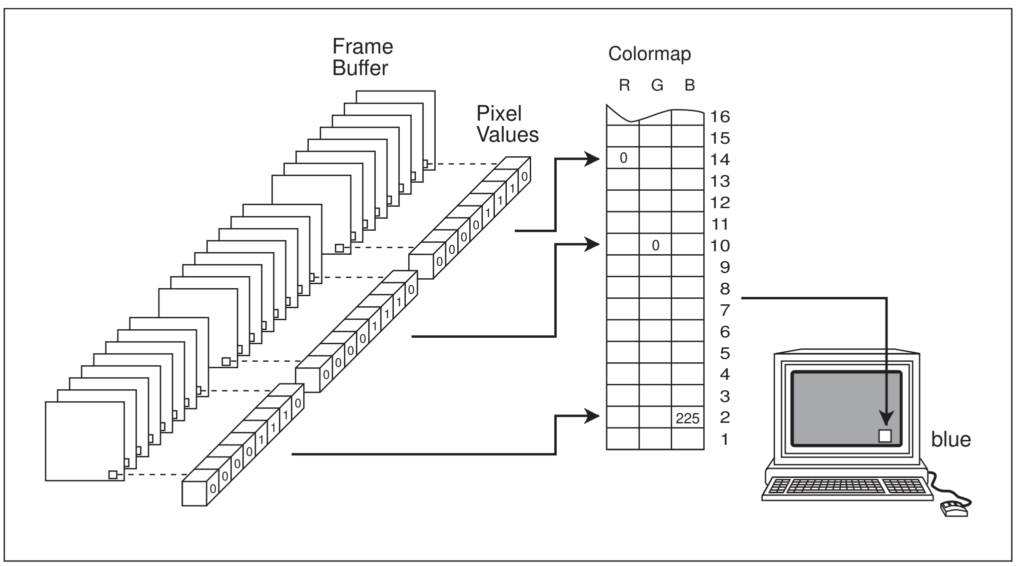


**Figure 11-1. Multiple planes used to index a colormap**

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 possible on the display is a function of the number of bits available in the colormap for RGB specification. If 8 bits are available for each primary, then the range of possible colors is $256^3$ (more than 16 million colors). This means that you can create incredibly precise differences between colors.

However, the number of different colors that can be displayed on the screen at any one time is a function of the number of planes. A four-plane system can index 2^4 colorcells (16 distinct colors); an 8-plane system can index $2^8$ colorcells (256 distinct colors); and a 24-plane system can index $2^{24}$ colorcells (more than 16 million distinct colors).

If you are using a 4-plane workstation, the fact that you can precisely define hundreds of different shades 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 or any mechanism for them to be selected even if they could be stored.

This limitation is made more significant by the fact that X is a multi-client environment. When X starts up, usually no colors are loaded into the colormap. As clients are invoked, certain of these cells are allocated. But when all of the free colorcells are used up, it is no longer possible to request new colors. When this happens, you will usually be given the closest possible color from those already allocated. However, you may instead be given an error message and told that there are no free colorcells.

In order 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 any of them. They can only be de-allocated by each application that uses them, and when all applications have de-allocated the cell, it is available for setting one again. Shared cells are most often used for background, border, and cursor colors.

Alternately, some clients have to 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. Such a program might have three bars of each primary color and a box that shows the mixed color. The primary bars would use shared cells, while the mixed color box would use a private cell.

In summary, some programs define colorcells to be read-only and sharable, while others define colorcells to be read/write and private.

To top it off, there are even clients that may temporarily swap in a private colormap of their own. If this happens, all other applications will be displayed in unexpected colors because of the way color is implemented. This is called *colormap flashing*, because the colormaps flash in and out as the pointer enters and leaves such clients. You can "lock" in the colormap of a client under the OpenWindows version of *olwm* using the Colormap Lock key (Ctrl/L2 on the Sun keyboard).

In order to minimize such conflicts, you should request precise colors only when necessary. By preference, use color names or hexadecimal specifications that you specified for other applications.

† There is a type of high-end display in which pixel values are used directly to control the illumination of the red, green, and blue phosphors. But far more commonly, the bits per pixel are used indirectly with the actual color values specified independently.

## 11.6 Border Width

Many clients accept a `-bw` option that is intended to specify the width of the window border in pixels. However, as in the case of the `-bd` (border color) option, under OPEN LOOK, this customization is generally useless because the *olwm* window frame effectively replaces most window borders.

As an alternative, you *can* change the width of the frame of all windows by specifying resources for *olwm* in a *.Xresources* or *.Xdefaults* file in your home directory. For more information, see Chapter 13, *Customizing olwm*, and the resources on the *olwm* reference page in Part Three of this guide.

## 11.7 xterm and cmdtool example

Here is an example that consolidates several of the options that we've discussed so far. We want a terminal emulator, with a window size of 80 columns by 48 rows (double the size of a conventional "24x80" ASCII terminal), at the left margin but with its top 100 pixels down from the top of the screen. And we want the name in its titlebar to be whatever is stored in the shell variable "HOSTNAME". If we prefer to use *xterm*, we could use this form:

```
xterm -geometry 80x48+0+100 -T "$HOSTNAME"&
```

If we are using *cmdtool* we could give this specification:

```
cmdtool -Ww 80 -Wh 40 -Wp 0 100 -Wl "$HOSTNAME"&
```

However, this attempt at an X Toolkit-compatible one:

```
cmdtool -geometry 80x48+0+100 -title "$HOSTNAME"&
```

will fail, because in this context the width and height are interpreted in pixels, so we get a very tiny window, about an inch high! By contrast, *xterm* interprets the geometry size in columns and rows, while *cmdtool* interprets it in pixels, and provides the alternate *-width* and *-height* command line arguments (columns and rows, respectively), so that you can specify it either way.

## 11.8 Summary

We've now shown most of the standard command line options for the Intrinsics toolkits (Athena and OLIT) as well as for the XView toolkit. Since color is so important in making a display visually appealing, we spent some time on it. In the next chapter we describe Resources, which are a way you can control the behavior of window clients without having to type your preferences each time you start that client.