

# COLOR

---

Color is the software for driving color displays. To run COLOR, you need either a Sun (3, 4 or SPARCstations) with CG4, CG6 color hardware and color display.

The machine independent color graphics code is stored in the library files `LLCOLOR.LCOM` and `COLOR.LCOM`. The Sun color driver resides in the file `MAIKOCOLOR.LCOM` which loads `LLCOLOR.LCOM`, `COLOR.LCOM`, and `TAKEBIGBM.LCOM`.

The CG4,CG6 device supports 8 bpp (bits per pixel) at a resolution of 1152 by 900.

Note: You cannot use COLOR if you are running Medley under X.

## Software Screens

---

To support the various hardware configurations and external display devices, the software has a special datatype, a "screen". A screen represents and controls a physical hardware display. It contains windows and icons, and tracks the mouse. There are two distinct types of screens: a black and white screen, and a color screen.

On workstations with a single hardware display, the display is shared by both the black and white screen and the color screen. It can be changed by moving the mouse cursor. The screen mode may be changed by moving the cursor out of the screen.

## Turning the Color Display Software On and Off

---

The color display software can be turned on and off. While the color display software is on, and a small amount of processing time is used to drive the color display.

(`COLORDISPLAYP`) [Function]

Returns `T` if the color display is on; otherwise it returns `NIL`.

(`COLORDISPLAYONOFF TYPE`) [Function]

Turns off the color display if `ONOFF` is set to `OFF`. If `ONOFF` is set to `ON`, it turns on the color-display-allocating memory for the color screen bitmap. `TYPE` should be `MAIKOCOLOR`, if you are using a Sun Workstation. The usual sequence of events is to `LOAD MAIKOCOLOR.LCOM` to drive GC4/GC6 color card and then to call `COLORDISPLAY` with the appropriate `TYPE` (`MAIKOCOLOR` for Sun Workstations) to turn the software on. For example,

```
(LOAD 'MAIKOCOLOR.LCOM)
(COLORDISPLAY 'ON 'MAIKOCOLOR)
```

Calling `COLORDISPLAY` allocates memory for the color screen bitmap in the `sysout`. Turning on the color display requires allocating the memory necessary to hold the color display screen bitmap. Turning off the color display does not free this memory; it still exists in the `sysout`.

## Colors

---

The number of bits per pixel (bpp) determines the number of different colors that can be displayed at one time. When there are 8 bpp, 256 colors can be displayed at once. A table, called a *color map*, determines what color actually appears for each pixel value. A color map gives the color in terms of how much of the three primary colors (red, green, and blue) is displayed on the screen for each possible pixel value.

A color can be represented as a number, an atom, or a triple of numbers. The color map provides the final interpretation of how much red, blue, and green a color represents. A color map maps a color number ( $[0 \dots 2^{n_{\text{bits}}}-1]$ ) into the intensities of the three color guns (primary colors red, green, and blue). Each entry in the color map has 8 bits for each of the primary colors, allowing 256 levels per primary or  $2^{24}$  possible colors (not all of which are distinct to the human eye). Within Xerox Lisp programs, colors can be manipulated as numbers, red-green-blue triples, names, or hue-lightness-saturation triples. Any function that takes a color accepts any of these different representations.

If a number is given, it is the color number used in the operation. It must be valid for the color bitmap used in the operation. (Since all of the routines that use a color need to determine the color's number, it is fastest to use numbers for colors. `COLORNUMBERP`, described below, provides a way to translate into numbers from the other representations.)

The following sections describe other ways to represent colors.

### Red-Green-Blue Triples

A red-green-blue (RGB) triple is a list of three numbers, each of which can be between 0 and 255. The nine digits are divided as follows:

- First 3 digits = red intensity
- Second 3 digits = green intensity
- Third 3 digits = blue intensity

When an RGB triple is used, the current color map is searched to find the color with the correct intensities. If none is found, an error is generated. (That is, no attempt is made by the system to assign color numbers to intensities automatically.) An example of an RGB triple is (255 255 255), which gives the color white.

RGB

[Record]

A record defined as (RED GREEN BLUE); it can be used to manipulate RGB triples.

COLORNAMES

[Association list]

Maps names into colors. The CDR of the color name's entry is used as the color corresponding to the color name. This can be any of the other representations.

Note: It can even be another color name. The system does not check for loops in the name space, such as would be caused by putting ' (RED . CRIMSON) and ' (CRIMSON . RED) on COLORNAMES.

Some color names are available in the initial system and allow color programs written by different users to coexist. These are:

Name	RGB	Number in default color maps	
BLACK	(0 0 0)	15	255
BLUE	(0 0 255)	14	252
GREEN	(0 255 0)	13	227
CYAN	(0 255 255)	12	224
RED	(255 0 0)	3	31
MAGENTA	(255 0 255)	2	28
YELLOW	(255 255 0)	1	3
WHITE	(255 255 255)	0	0

## Hue–Lightness–Saturation Triples

A hue–lightness–saturation triple is a list of three numbers:

- The first number (HUE) is an integer between 0 and 355. It indicates a position in degrees on a color wheel (blue at 0, red at 120, and green at 240).
- The second (LIGHTNESS) is a real number between zero. It indicates how much total intensity is in the color.
- The third (SATURATION) is a real number between zero and one. It indicates how disparate the three primary levels are.

HLS

[Record]

A record defined as (HUE LIGHTNESS SATURATION); it is provided to help you manipulate HLS triples.

Example: the color blue is represented in HLS notation by (0 .5 1.0).

(COLORNUMBERP *COLOR* *BITSPERPIXEL* *NOERRFLG*)

[Function]

Returns the color number (offset into the screen color map) of *COLOR*. *COLOR* can be one of the following:

- A positive number less than the maximum number of colors
- A color name
- An RGB triple
- An HLS triple

If *COLOR* is one of the above and is found in the screen color map, its color number in the screen color map is returned. If not, an error is generated unless *NOERRFLG* is non-NIL, in which case NIL is returned.

(RGBP *X*)

[Function]

Returns *X* if *X* is an RGB triple; otherwise it returns NIL.

(HLSP *X*)

[Function]

Returns *X* if *X* is an HLS triple; otherwise it returns NIL.

## Color Maps

---

The screen color map holds the information about what color is displayed on the color screen for each pixel value in the color screen bitmap. If you change the values in the current screen color map, this change is reflected in the colors displayed at the next vertical retrace (approximately 1/30 of a second). The color map can be changed to obtain dramatic effects.

(SCREENCOLORMAP *NEWCOLORMAP*) [Function]

Reads and sets the color map used by the color display. If *NEWCOLORMAP* is non-NIL, it should be a color map, and SCREENCOLORMAP sets the system color map to be that color map. The value returned is the value of the screen color map before SCREENCOLORMAP was called. If *NEWCOLORMAP* is NIL, the current screen color map is returned without change.

(CMYCOLORMAP *CYANBITS MAGENTABITS YELLOWBITS  
BITSPERPIXEL*) [Function]

Returns a color map that assumes the *BITSPERPIXEL* bits are to be treated as three separate color planes: *CYANBITS* bits in the cyan plane, *MAGENTABITS* bits in the magenta plane, and *YELLOWBITS* bits in the yellow plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 0 and black is 255.

(RGBCOLORMAP *REDBITS GREENBITS BLUEBITS  
BITSPERPIXEL*) [Function]

Returns a color map that assumes the *BITSPERPIXEL* bits are to be treated as three separate color planes: *REDBITS* bits in the red plane, *GREENBITS* bits in the green plane, and *BLUEBITS* bits in the blue plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 255 and black is 0.

(GRAYCOLORMAP *BITSPERPIXEL*) [Function]

Returns a color map containing shades of gray. White is 0 and black is 255.

(COLORMAPCREATE *INTENSITIES BITSPERPIXEL*) [Function]

Creates a color map for a screen that has *BITSPERPIXEL* bits per pixel. If *BITSPERPIXEL* is NIL, the number of bits per pixel is taken from the current color display setting. *INTENSITIES* specifies the initial colors that should be in the map. If *INTENSITIES* is not NIL, it should be a list of color specifications other than color numbers, e.g., the list of RGB triples returned by the function INTENSITIESFROMCOLOR MAP.

(INTENSITIESFROM COLORMAP *COLORMAP*) [Function]

Returns a list of the intensity levels of *COLORMAP* in a form accepted by COLORMAPCREATE. The default is SCREENCOLORMAP. This list can be written on file, providing a way of saving color map specifications.

(COLORMAPCOPY *COLORMAP BITSPERPIXEL*) [Function]

Returns a color map that contains the same color intensities as *COLORMAP* if *COLORMAP* is a color map. Otherwise, it returns a color map with default color values.

(MAPOFACOLOR *PRIMARIES*)

[Function]

Returns a color map that is different shades of one or more of the primary colors. For example, (MAPOFACOLOR ' (RED GREEN BLUE)) gives a color map of different shades of gray; (MAPOFACOLOR ' RED) gives different shades of red.

## Changing Color Maps

The following functions are provided to access and change the intensity levels in a color map.

(SETCOLORINTENSITY *COLORMAP COLORNUMBER COLORSPEC*)

[Function]

Sets the primary intensities of color number *COLORNUMBER* in the color map *COLORMAP* to the ones specified by *COLORSPEC*. *COLORSPEC* can be either an RGB triple, an HLS triple, or a color name. The value returned is NIL.

(COLORLEVEL *COLORMAP COLORNUMBER PRIMARYNEWLEVEL*)

[Function]

Sets and reads the intensity level of the primary color *PRIMARY* (RED, GREEN, or BLUE) for the color number *COLORNUMBER* in the color map *COLORMAP*. If *NEWLEVEL* is a number between 0 and 255, it is set. The previous value of the intensity of *PRIMARY* is returned.

(ADJUSTCOLORMAP *PRIMARY DELTA COLORMAP*)

[Function]

Adds *DELTA* to the intensity of the *PRIMARY* color value (RED, GREEN, or BLUE) for every color number in *COLORMAP*.

(ROTATECOLORMAP *STARTCOLOR THRUCOLOR*)

[Function]

Rotates a sequence of colors in the SCREENCOLORMAP. The rotation moves the intensity values of color number *STARTCOLOR* into color number *STARTCOLOR*+1, the intensity values of color number *STARTCOLOR*+1 into color number *STARTCOLOR*+2, etc., and *THRUCOLOR*'s values into *STARTCOLOR*.

(EDITCOLORMAP *VAR NOQFLG*)

[Function]

Allows interactive editing of a color map. If *VAR* is an atom whose value is a color map, its value is edited. Otherwise a new color map is created and edited. The color map being edited becomes the screen color map while the editing takes place so that its effects can be observed. The edited color map is returned as the value. If *NOQFLG* is NIL and the color display is on, you are asked if you want a test pattern of colors. If you answer "yes," the function SHOWCOLORTESTPATTERN is called, which displays a test pattern showing blocks of each of the possible colors.

The system prompts for the location of a color control window to be placed on the black-and-white display. This window allows the value of any of the colors to be changed. The number of the color being edited is in the upper left part of the window. Six bars are displayed. The right three bars give the color intensities for the three primary colors of the current color number. The left three bars give the value of the color's Hue, Lightness, and Saturation parameters. To change these levels, position the mouse cursor in one of the bars and press the left mouse button. While the left button is down, the value of that parameter tracks the Y position of the cursor. When the left button is released, the color tracking stops. To change the color being edited, press the middle

mouse button while the cursor is inside the edit window. This brings up a menu of color numbers. Select one to set the current color to the selected color.

The color being edited can also be changed by selecting the menu item PickPt. This switches the cursor onto the color screen and allows you to select a point from the color screen. It then edits the color of the selected point.

To stop the editing, move the cursor into the title of the editing window and press the middle button. This brings up a menu. Select Stop to quit.

## Color Bitmaps

---

A color bitmap is actually a bitmap that has more than one bit per pixel. Use the function `BITSPERPIXEL` to test whether a bitmap is a color bitmap.

(`BITSPERPIXEL` *BITMAP*) [Function]

Returns the bits per pixel of *BITMAP*. If this does not equal one, *BITMAP* is a color bitmap.

In multiple-bit-per-pixel bitmaps, the bits that represent a pixel are stored contiguously. `BITMAPCREATE` is passed a *BITSPERPIXEL* argument to create multiple-bit-per-pixel bitmaps.

With `CG4,CG6`, `BITSPERPIXEL` is 8 for color bitmaps.

(`BITMAPCREATE` *WIDTH HEIGHT BITSPERPIXEL*) [Function]

Creates a color bitmap that is *WIDTH* pixels wide by *HEIGHT* pixels high allowing *BITSPERPIXEL* bits per pixel. Currently any value of *BITSPERPIXEL* except 1, 4, 8, or `NIL` (defaulting to 1) causes an error.

`BITMAPCREATE` may return two types of bitmap. An ordinary `BITMAP` type usually contains up to 131066 cells (32 bits of data) in it. A larger bitmap can be created as `BIGBM`.

(`DATATYPE BIGBM (BIGBMWIDTH BIGBMHEIGHT BIGBMLIST)`)

`BIGBM` has the list of pointers which point to `BITMAPS`.

An 8 bpp screen bitmap (1052 \* 900) uses approximately 1 Mbyte.

(`COLORSCREENBITMAP`) [Function]

Returns the bitmap that is being or will be displayed on the color display. This is `NIL` if the color display has never been turned on (see `COLORDISPLAY` below).

## Screens, Screenpositions, and Screenregions

---

In addition to positions and regions, you need to be aware of screens, screenpositions, and screenregions in the presence of multiple screens.

## Screens

SCREEN [Datatype]

There are generally two screen datatype instances in existence when working with color. This is because you are attached to two displays, a black and white display and a color display.

(MAINSCREEN) [Function]

Returns the screen datatype instance that represents the black and white screen. This will be something like {SCREEN}#74,24740.

(COLORSCREEN) [Function]

Returns the screen datatype instance that represents the color screen. Screens appear as part of screenpositions and screenregions, serving as the extra information needed to make clear whether a particular position or region should be viewed as lying on the black and white display or the color display.

(SCREENBITMAP *SCREEN*) [Function]

Returns the bitmap destination of *SCREEN*. If *SCREEN* is NIL, it returns the black and white screen bitmap.

## Screenpositions

SCREENPOSITION [Record]

Somewhat like a position, a screenposition denotes a point in an X,Y coordinate system on a particular screen. Screenpositions are defined according to the following record declaration:

```
(RECORD SCREENPOSITION (SCREEN . POSITION)
  (SUBRECORD POSITION))
```

A SCREENPOSITION is an instance of a record with fields XCOORD, YCOORD, and SCREEN and is manipulated with the standard record package facilities. For example, (create SCREENPOSITION XCOORD \_ 10 YCOORD \_ 20 SCREEN \_ (COLORSCREEN)) creates a screenposition representing the point (10,20) on the color display. You can extract the position of a screenposition by fetching its POSITION. For example, (fetch (SCREENPOSITION POSITION) of SP12).

## Screenregions

SCREENREGION [Record]

Similar to a region, a screenregion denotes a rectangular area in a coordinate system. Screenregions are defined according to the following record declaration:

```
(RECORD SCREENREGION (SCREEN . REGION) (SUBRECORD REGION))
```

A screenregion is characterized by the coordinates of its bottom left corner and its width and height. A SCREENREGION is a record with fields LEFT, BOTTOM, WIDTH, HEIGHT, and SCREEN. It can be manipulated with the standard record package facilities. There are access functions for the REGION record that return the TOP and RIGHT of the region. You can extract the region of a screenregion by fetching its REGION. For example, (fetch (SCREENREGION REGION) of SR8).

## Screenposition and Screenregion Prompting

The following functions can be used by programs to allow you to interactively specify screenpositions or screenregions on a display screen.


(GETSCREENPOSITION *WINDOW CURSOR*) [Function]

Similar to GETPOSITION. Returns a SCREENPOSITION you specify. GETSCREENPOSITION waits for you to press and release the left button of the mouse and returns the cursor screenposition at the time of release. If *WINDOW* is a WINDOW, the screenposition is on the same screen as *WINDOW* and in the coordinate system of *WINDOW*'s display stream. If *WINDOW* is NIL, the position is in screen coordinates.

(GETBOXSCREENPOSITION *BOXWIDTH BOXHEIGHT ORGX ORGY WINDOW PROMPTMSG*) [Function]

Similar to GETBOXPOSITION. Returns a SCREENPOSITION you specified. This function allows you to position a "ghost" region of size *BOXWIDTH* by *BOXHEIGHT* on a screen, and returns the SCREENPOSITION of the lower left corner of the screenregion chosen. A ghost region is locked to the cursor so that if the cursor is moved, the ghost region moves with it. To change to another corner, press and hold the right button. With the right button down, the cursor can be moved across a screen or to other screens without effect on the ghost region frame. When the right button is released, the mouse snaps to the nearest corner, which then becomes locked to the cursor. (To change the held corner after the left or middle button is down, hold both the original button and the right button down while moving the cursor to the desired new corner, then release just the right button.) When the left or middle button is pressed and released, the lower left corner of the screenregion chosen at the time of release is returned. If *WINDOW* is a WINDOW, the screenposition is on the same screen as *WINDOW* and in the coordinate system of *WINDOW*'s display stream. If *WINDOW* is NIL, the position is in the screen.


(GETSCREENREGION *MINWIDTH MINHEIGHT OLDREGION NEWREGIONFN NEWREGIONFNARG INITCORNERS*) [Function]

Similar to GETREGION. Returns a SCREENREGION you specified. This function allows you to specify a new screenregion and returns that screenregion. GETSCREENREGION prompts for a screenregion by displaying a four-pronged box next to the cursor arrow at one corner of a "ghost" region: . If you press the left button, the corner of a "ghost" screenregion opposite the cursor is locked where it is. Once one corner has been fixed, the ghost screenregion expands as the cursor moves.

To specify a screenregion:

1. Move the ghost box so that the corner opposite the cursor is at one corner of the intended screenregion.
2. Press the left button.
3. Move the cursor to the screenposition of the opposite corner of the intended screenregion while holding down the left button
4. Release the left button.



Before one corner has been fixed, you can switch the cursor to another corner of the ghost screenregion by holding down the right button. With the right button down, the cursor changes to a "forceps"  and the cursor can be moved across a screen or to other screens without effect on the ghost screenregion frame. When the right button is released, the cursor snaps to the nearest corner of the ghost screenregion.

After one corner has been fixed, you can still switch to another corner. To change to another corner, continue to hold down the left button and hold down the right button also. With both buttons down, the cursor can be moved across a screen or to other screens without effect on the ghost screenregion frame. When the right button is released, the cursor snaps to the nearest corner, which becomes the moving corner. In this way, the screenregion may be moved all over a screen, and to other screens, before its size and screenposition are finalized.

The size of the initial ghost screenregion is controlled by the *MINWIDTH*, *MINHEIGHT*, *OLDREGION*, and *INITCORNERS* arguments.

(GETBOXSCREENREGION *WIDTH HEIGHT ORGX ORGY WINDOW PROMPTMSG*)  
[Function]

Similar to GETBOXREGION. Returns a SCREENREGION you specified. This function performs the same prompting as GETBOXSCREENPOSITION and returns the SCREENREGION specified instead of the SCREENPOSITION of its lower left corner.

## Color Windows and Menus

The Medley window system provides both interactive and programmatic constructs for creating, moving, reshaping, overlapping, and destroying windows in such a way that a program can use a window in a relatively transparent fashion. Menus are a special type of window provided by the window system, used for displaying a set of items, on which you use the mouse cursor to make a selection. The menu facility also allows you to create and use menus in interactive programs.

(CREATEW *REGION TITLE BORDERSIZE NOOPENFLG*) [Function]

Creates a new window. *REGION* indicates where and how large the window should be by specifying the exterior screenregion of the window. In a user environment with multiple screens, such as a black and white screen and color screen both connected to the same machine, there is a new special problem in indicating which screen the *REGION* is supposed to be a region of. To resolve this problem, allow CREATEW to take screenregion arguments as *REGION*.

For example:

```
(SETQ FOO (CREATEW (CREATE SCREENREGION
  SCREEN _ (COLORSCREEN)
  LEFT _ 20  BOTTOM _ 210
  WIDTH _ 290 HEIGHT _ 170)
  "FOO WINDOW"))
```

creates a window titled "FOO WINDOW" on the color screen. To create a window on the black and white screen, use SCREEN \_ (MAINSCREEN) in the CREATE SCREENREGION expression.

Note: It is still perfectly legal to pass in a *REGION* that is a region, not a screenregion, to `CREATEW`, but it is better to pass screenregions. When *REGION* is a region, *REGION* is disambiguated in a somewhat arbitrary manner that may not always turn out to be what you want.

When *REGION* is a region, *REGION* is disambiguated by coercing *REGION* to be a screenregion on the screen which currently contains the cursor. Software calling `CREATEW` with regions instead of screenregions tends to do what you expect.

(`WINDOWPROP WINDOW PROP NEWVALUE`)

[NoSpread Function]

If *PROP* is ' `SCREEN`, then `WINDOWPROP` returns the screen *WINDOW* is on. If *NEWVALUE* is given (even if given as `NIL`), with *PROP* is ' `SCREEN`, then `WINDOWPROP` generates an error. Any other *PROP* name is handled in the usual way.

(`OPENWINDOWS SCREEN`)

[Function]

Returns a list of all open windows on *SCREEN* if *SCREEN* is a screen datatype such as (`MAINSCREEN`) or (`COLORSCREEN`). If *SCREEN* is `NIL`, then *SCREEN* defaults to the screen containing the cursor. If *SCREEN* is `T`, a list of all open windows on all screens is returned.

---

## Fonts in Color

There is no special function for creating color fonts. You use the same font on every screen.

You can use color characters by specifying the foreground color (the color of the characters of the font) and background color (the color of the background behind the characters that gets printed along with the characters) in the `DISPLAYSTREAM` data used for the current window (see below).

---

## Using Color

The current color implementation allows display streams to operate on color bitmaps. The two functions `DSPCOLOR` and `DSPBACKCOLOR` set the color in which a stream draws when the user defaults a color argument to a drawing function and printing characters.

(`DSPCOLOR COLOR STREAM`)

[Function]

Sets the foreground color of a stream. It returns the previous foreground color. If *COLOR* is `NIL`, it returns the current foreground color without changing anything. The default foreground color is `MINIMUMCOLOR=0`, which is white in the default color maps.

(`DSPBACKCOLOR COLOR STREAM`)

[Function]

Sets the background color of a stream. It returns the previous background color. If *COLOR* is `NIL`, it returns the current background color without changing anything. The default background color is (`MAXIMUMCOLOR BITSPERPIXEL`) = 255, which is black in the default color maps.

The BITBLT, line-drawing and curve-drawing routines know how to operate on a color-capable stream. Following are some notes about them.

## BITBLTing in Color

If you are BITBLTing from a color bitmap onto another color bitmap of the same bpp, the operations PAINT, INVERT, and ERASE are done on a bit level, not on a pixel level. Thus, painting color 3 onto color 10 results in color 11.

If BITBLTing from a black-and-white bitmap onto a color bitmap, the 1 bits appear in the DSPCOLOR, and the 0 bits in DSPBACKCOLOR. BLTing from black-and-white to color is fairly expensive; if the same bitmap is going to be put up several times in the same color, it is faster to create a color copy and then to BLT the color copy.

If the source type is TEXTURE and the destination bitmap is a color bitmap, the *Texture* argument is taken to be a color. Therefore, to fill an area with the color BLUE assuming COLORSTR is a stream whose destination is the color screen, use (BITBLT NIL NIL NIL COLORSTR 50 75 100 200 'TEXTURE 'REPLACE 'BLUE) .

## Drawing Curves and Lines in Color

For the functions DRAWCIRCLE, DRAWELLIPSE, and DRAWCURVE, the notion of a brush has been extended to include a color. A BRUSH is now (BRUSHSHAPE BRUSHSIZE BRUSHCOLOR). Also, a brush can be a bitmap (which can be a color bitmap).

Line-drawing routines take a color argument which is the color of the line if the destination of the display stream is a color bitmap.

(DRAWLINE *X1 Y1 X2 Y2 WIDTH OPERATION STREAM*  
*COLOR*) [Function]

(DRAWTO *X Y WIDTH OPERATION STREAM COLOR*) [Function]

(RELDRAWTO *X Y WIDTH OPERATION STREAM COLOR*) [Function]

(DRAWBETWEEN *POS1 POS2 WIDTH OPERATION STREAM*  
*COLOR*) [Function]

If the *COLOR* argument is NIL, the DSPCOLOR of the stream is used.

## Printing in Color

Printing only works in REPLACE mode. The characters have a background color and a foreground color determined by the DSPCOLOR and DSPBACKCOLOR .

Example of printing to an 8 bpp color screen:

```
(SETQ FOO (CREATEW (CREATE SCREENREGION
  SCREEN _ (COLORSCREEN) LEFT _ 20
  BOTTOM _ 210 WIDTH _ 290
  HEIGHT _ 170) "FOO WINDOW"))
(DSPCOLOR FOO 'GREEN)
(DSPBACKCOLOR 'YELLOW)
```

(PRINT 'HELLO FOO) prints in green against a yellow background.

## Operating the Cursor on the Color Screen

---

To move the cursor to the color screen, slide the cursor off the left or right edge of the black and white screen onto the color screen, or call the `CURSORPOSITION` or `CURSORSCREEN` function.

(WORPCURSOR *ENABLE*) [Function]

If *ENABLE* is `NIL`, you cannot exit the current screen by sliding the cursor off.

(CURSORPOSITION *NEWPOSITION* - -) [Function]

*NEWPOSITION* can be a position or a screenposition.

(CURSORSCREEN *SCREEN XCOORD YCOORD*) [Function]

Moves the cursor to the screenposition determined by *SCREEN*, *XCOORD*, and *YCOORD*. *SCREEN* should be the value of either `(COLORSCREEN)` or `(MAINSCREEN)`.

While on the color screen, the cursor is placed by doing `BITBLTs` in software rather than with microcode and hardware as with the black and white cursor. It is automatically taken down whenever an operation is performed that changes any bits on the color screen. The speed of the color cursor compares well with that of the black-and-white cursor but there can be a noticeable flicker when there is much input/output to the color screen. While the cursor is on the color screen, the black-and-white cursor is cleared giving the appearance that there is never more than one cursor at a given time.

## Miscellaneous Color Functions

---

(COLORIZEBITMAP *BITMAP 0COLOR 1COLOR BITSPERPIXEL*) [Function]

Creates a color bitmap from a black-and-white bitmap. The returned bitmap has color number *1COLOR* in those pixels of *BITMAP* that were 1 and *0COLOR* in those pixels of *BITMAP* that were 0. This provides a way of producing a color bitmap from a black-and-white bitmap.

(UNCOLORIZEBITMAP *BITMAP COLORMAP*) [Function]

Creates a black-and-white bitmap from a color bitmap.

(SHOWCOLORTESTPATTERN *BARSIZE*) [Function]

Displays a pattern of colors on the color display. This is useful when editing a color map. The pattern has squares of the 16 possible colors laid out in two rows at the top of the screen. Colors 0 through 7 are in the top row, and colors 8 through 15 are in the next row. The bottom part of the screen is filled with bars of *BARSIZE* width with consecutive color numbers. The pattern is designed so that every color has a border with every other color (unless *BARSIZE* is too large to allow room for every color—about 20).

[This page intentionally left blank]