The Figaro X11 Display Server
Users' and Programmers' Guide

Sam Southard, Jr.

March 20, 1993

Contents

1 Introduction

The Figaro X11 display server (figdisp) is designed to provide all features truly necessary in a display while being as fast as possible. Since all the nicest features are worthless if the display is slow, the primary emphasis is on speed. Of course, "fast" depends on the CPU, memory configuration, etc. of the machine on which you are running Figaro. Figdisp is adequate on a SPARCstation with at least 16 MB of memory. Figdisp requires an 8-bit visual for bitmap graphics (line graphics can use any depth visual). Most displays which have more than 8 bits (the Sun 24-bit GS board, for example) support 8-bit visuals, so figdisp will work on those displays as well. Pgdisp, the line-graphics only version of figdisp, will work on any visual (including read-only visuals).

Figdisp is compatible with both public domain X11 and OpenWindows. Figdisp is known to work on the following configurations:

- Sun 4 series (cgtwo, cgsix, cgtwelve, and bwtwo) running OpenWindows 2.0 or 3.0 with the olwm window manager (all visuals for pgdisp, all 8-bit visuals for figdisp).
- NCD 14c (8-bit color) running xdm with the olwm window manager (all visuals for pgdisp, all 8-bit visuals for figdisp).

If you have another configuration on which pgdisp or figdisp runs, please send a message to sns@deimos.caltech.edu (Internet), sns@citdeimo (Bitnet), or DEIMOS::SNS (DECnet/Span).

2 Users' Guide

2.1 Starting Figdisp

Before starting figdisp, both the Figaro and the X11 environment must be set up. The X11 environment consists of setting the DISPLAY environment variable appropriately. For example:

```
% setenv DISPLAY :0
```

or

```
% setenv DISPLAY lo-fan:0.0
```

The first example would cause the display to appear on the local machine. Other values which can be used to accomplish this are "unix:0.0" and "lhost:0.0", where lhost is the name of the local host.

The second example would cause the display to appear on the machine "lo-fan," which could be the either local machine or another machine running an X11 server.[1] See the X11 documentation for a more detailed description of DISPLAY. Multiple copies of figdisp/pgdisp may be running at any given time. The only limit is your machine's memory.

Once the display environment variable set, simply typing the line

```
% figdisp [options] &
```

or

[1]The remote machine can be any machine which is running an X11R4 server, not just a Sun.

```
% pgdisp [options] &
```

will start the display server. The options are optional and are discussed in Section 2.3.1, **Customizing Figdisp**. If you are using Figaro, the "&" is not necessary - figdisp is aliased to do the correct thing. If you wish to start up a second copy of figdisp or pgdisp, type the line

```
% figdisp -id # [options] &
```

or

```
% pgdisp -id # [options] &
```

represents any integer. Note that there can not be a copy of figdisp and a copy of pgdisp running on the same screen with the same id.

The Figaro environment is initially set up by the figlnm.com and figaro.com scripts.[2] Figdisp is the default display for TVPCKG on Suns, so if you only use TVPCKG routines you do not need to do anything else. If for some reason this gets changed, the command

```
% setenv TV_DEVICE FIGDISP
```

will tell TVPCKG to use figdisp.

If you have multiple copies of figdisp running, the FIGDISP environment variable, 0 by default, determines which one will be used. If you wish to use a different copy of figdisp, use the command

```
% setenv FIGDISP #
```

should be the same as one of the id numbers used to start a copy of figdisp.

If the routines you want to use PGPLOT to do line graphics, you will need use the SOFT command to tell PGPLOT to use figdisp. For example:

```
% soft /xdisp
```

or

```
% soft /xd
```

The full name of the PGPLOT device which corresponds to figdisp (or pgdisp) is '/xdisp', as shown by the first example. The second example shows the minimum string necessary to cause line graphics output to appear in figdisp. You must have already set up the figaro environment (using the figlnm.com and figaro.com scripts) before running SOFT. If you wish to use a copy of pgdisp other than the default, use the command

```
% soft #/xdisp
```

should be the same as one of the id numbers used to start a copy of pgdisp or figdisp.

[2]See the Sun Figaro release notices for details on setting up the Figaro environment.

2.2 Using Figdisp

Figdisp has several output windows: one for bitmap graphics, one for line graphics, one for displaying data values (called the “patch” window), one for displaying where the visible portion of the image fits into the total image (called the “location” window), one for displaying the current color map, and one for displaying the FWHM and center of a specified point (called the “seeing” window). The pgdisp program is identical to figdisp, except that it only supports line graphics. The line graphics window does not appear until it is necessary, but the bitmap graphics window is always displayed. When iconized, the bitmap graphics window is labeled “bitmap” and the line graphics window is labeled “figdisp” if the program running is figdisp and “pgdisp” if the program running is pgdisp.

Table 1 lists the keys and mouse buttons used to access the various features. Note that these are only defaults — the key to access any function can be configured by the user. The default set of keys is based on the Sun keyboard — L1 is the same as F11 and R1 is the same as F21.

When figdisp is started it prints out a short message informing the user of the help key and referring him to the manual for information about the various ways to customize figdisp. This message can be disabled.[3]

2.2.1 The Bitmap Window

The bitmap graphics window was designed to make common image manipulation tasks easily accessible. It supports may functions, all of which are available from the mouse or keyboard; there are no menus. This window can be resized arbitrarily, without relation to the image size. If the display server is actually doing something (such as reading a new image) while the user resizes the image, the image will look odd until figdisp gets a chance to take care of the resize. The initial size is determined by the figdisp.bm.geometry resource.

The portion of the image visible in the bitmap window can be changed with the left mouse button. Pressing it will move the point under the cursor to the center of the window.

The X and Y zoom factors can be increased, decreased, and reset (F2 (controlled by the figdisp.zoomIn resource), F4 (figdisp.zoomOut), and F3 (figdisp.zoomNorm)) together. They can also be increased or decreased independently (R7 (figdisp.x.zoomIn) and R9 (figdisp.x.zoomOut) for X, R13 (figdisp.y.zoomIn) and R15 (figdisp.y.zoomOut) for Y). It is not possible to use zoom, resize, pan, or any combination of them to cause the entire image to go out of the displayed area.

Pressing F5 (figdisp.help) produces a message describing the function of each of the buttons and keys, as well some general information about the display server. The message is updated to reflect any changes the user has made to the key interpretations.

Pressing F6 (figdisp.cursor) causes this window's title to change to display the cursor location in the image, the data value at that location, and the X and Y zoom factors (if not 1). This information is updated as the cursor is moved. Pressing F6 again causes the window's title to return to the default.

Pressing F7 (figdisp.recenter) moves the point at the center of the image to the center of the window.

Pressing F8 (figdisp.showLoc) causes the “location” window to appear. Pressing F8 again causes the location window to disappear.

Pressing F10 (figdisp.showCmap) causes the color map window to appear. Pressing F10 again causes the color map window to disappear.

Pressing F9 (figdisp.quit) causes the display server to exit.

[3] See the figdisp.showhelp resource.

Button/key	**Feature**
Left button	If in default “mouse mode”, pan the image. Otherwise, set the upper left corner of the image statistics box. If the shift key is pressed, the left mouse button manipulates the color look-up tables.
Middle button	Display an arbitrary line plot.
Right button	If in default “mouse mode”, manipulate the color look-up tables. Otherwize, set the lower right corner of the image statistics box.
F2	Increase X and Y zoom factors.
F3	Reset X and Y zoom factors to 1.
F4	Decrease X and Y zoom factors.
F5	Help.
F6	Track cursor
F7	Re-center
F8	Toggle the display of the location window.
F9	Quit.
F10	Toggle the display of the color map window.
L1	Display the patch window. Pressing this button again inside the patch window dismisses it.
L2	Display a row plot in the line graphics window. The way in which this plot is generated (left to right, etc) can be configured by the user.
L3	Print a copy of the image.
L4	Print a copy of the portion of the image visible on the main screen.
L6	Inhibit keystroke interpretations until L6 is pressed again.
L8	Invert the color look-up tables.
L9	Calculate the center and FWHM of a star near the cursor.
L10	Display a column plot in the line graphics window. The way in which this plot is generated (left to right, etc) can be configured by the user.
,	Decrease the line plot line width.
.	Increase the line plot line width.
/	Reset the number of pixels averages together to one (only the central pixel is used).
¡	Decrease the number of times the LUT is wrapped.
¿	Increase the number of times the LUT is wrapped.
?	Reset to no LUT wrapping.
R1	Cycle between linear, histogram equalization, and limited linear color map allocation methods.
R2	Toggle between the default mouse mode and the mode where the left and right mouse buttons control the corners of the image statistics box.
R3	Update the image statistics window.
R7	Increase the X zoom factor.
R9	Decrease the X zoom factor.
R13	Increase the Y zoom factor.
R15	Decrease the Y zoom factor.

Table 1: Figdisp Commands

Pressing L1 (figdisp.showPatch) causes the "patch" window, which contains the numerical values of the points around the cursor, to appear. Pressing L1 again updates the points in the patch window to reflect the new cursor position. Pressing L1 inside the patch window causes it to disappear.

Pressing L3 (figdisp.imagePs) produces a PostScript copy of the entire image. If the figdisp.psFile resource is set to "direct" (the default), the output goes directly to the printer specified by the figdisp.printer resource. If the figdisp.printer resource is set to "PRINTER" the output is sent to the printer specified by the environment variable PRINTER. Otherwise, it is sent to the specified printer. If the figdisp.psFile resource is not set to direct, a file of the form "figdisp.psFile-#" is produced, where # is a sequence number. For example, if figdisp.psFile is spectra.ps, the output file of the first L3 press will be in spectra.ps-0, the second will be in spectra.ps-1, etc. Figdisp.psFile set to something other than direct takes precedence over the setting of figdisp.printer. Note that the file specification is relevant to the directory in which figdisp was started.

Pressing L4 (figdisp.windowPs) produced a PostScript copy of the portion of the image visible in the bitmap graphics window. The same resources as in figdisp.imagePs are used to specify the output destination.

Pressing L6 (figdisp.inhibit) inhibits the interpretation of all keystrokes until L6 is pressed again. This is useful if you have changed the keys figdisp uses to perform various functions and a client program needs to see that keystroke. For example, if you change the figdisp.quit key to be "q" and the program (ICUR, for example) needs to see a "q", you would press the sequence "L6qL6".

Pressing L9 (figdisp.showSee) near a star calculates the X and Y center and FWHM of the star and displays it in the "seeing" window. Pressing L9 again in the bitmap window updates the contents of the seeing window to reflect the new cursor position. Pressing L9 inside the seeing window causes it to disappear.

2.2.2 Manipulating the Color Maps

Figdisp allows the user to manipulate the color map entries in several ways. Pressing L8 (figdisp.invertCmap) inverts the color map.

Pressing and holding the right mouse button (assuming the mouse buttons are in the default mode) allows the user to dynamically change the color map. The X position in the bitmap window controls the offset into the color look-up tables and the Y position controls the slope. The algorithm used is identical to the Figaro TVTRANS routine (used by the CPOS T function). Releasing the right button within 10 pixels of the point where it was depressed causes the offset and slope to be reset.

Pressing and holding the left mouse button while the shift key is held down changes the offset added to the data value to get a color map window. Only the X position of the cursor is significant for these manipulations. For example, if this offset is set to 20 a pixel value of 2 would use the LUT entry for pixel 22. In contrast to the offset and contrast manipulation controlled with the right mouse button, the color map table is assumed to wrap around, so that, with an offset of 20 and an 8-bit image, a pixel value of 250 would use the LUT entry for pixel 14. Releasing the left mouse button within 10 pixels of the point where it was depressed resets this offset.

The next few manipulations are mainly useful with 16-bit images, or if you have very few colors available to figdisp. Pressing "R1" (figdisp.histogram) cycles between three methods for allocating LUT entries. The default is a simple linear mapping. Next is a histogram equalization mode, where an LUT entries are distributed among the pixels according to how frequently the pixels appear in the image. The third method is a linear mapping with the lowest and highest 2.5% of the pixel values excluded from the mapping (they get mapped to the lowest and highest LUT entry).

The latter two methods (histogram equalization and a partial linear mapping) calculate a histogram of some of the pixels in the image. By default, all pixels in the image are used. If you have large images, this can be somewhat slow (although the histogram is only calculated once for a given image). The figdisp.histgeometry resource controlls the area of the image used to calculate the histogram. If the X dimension of the given geometry is greater than 0 and less than the image width, the area used to calculate this histogram is centered on the image center and has the given width. A similar test applies to the given Y dimension.

You can also have each color map wrap around multiple times. Pressing "¿" (figdisp.increaseLUTWrap) increases the number of times the color map is, wrapped, pressing "¡" (figdisp.decreaseLUTWrap) increases it, and "?" (figdisp.resetLUTWrap) resets it to no wrapping. Changing the number of times the color map is wrapped automatically switches the color map allocation to linear if it was in histogram equalization or limited linear modes. The figdisp.initLUTWrap resource controls the initial number of times the LUT will be wrapped.

All of these manipulation methods may be combined in any way. All changes are reflected in the color map window.

2.2.3 Line Plots

Figdisp has the ability to take a row, column, or arbitrary line in the bitmap window and produce a plot of the data values along that line. In addition to simply using the points along a line, figdisp uses a line width to simulate a slit in a spectrograph. All the data values at a given point along the line (for example, all the values in a column of the specified width in a row plot) are averaged to produce a data point. All labels are in the font specified by the figdisp.font resource.

Plots can either be made in a point-to-point fashion (the default), or in histogram form. The figdisp.line.histogram resource and -

no

plothist command ine arguments control this behavior.

The line width is decreased by pressing "," (figdisp.decreaseSlit), increased by pressing "." (figdisp.increaseSlit), and reset to 1 by pressing "/" (figdisp.resetSlit). The line width starts out as 1 (only the central pixel is used). Each increase causes the number of pixels averaged together, not including the central pixel, to go up by a power of two. For example, the line width starts out at 1. The first increase makes the line width 3 ($2^1 + 1$), the second increase makes the line width 5 ($2^2 + 1$), and so on. Decreasing the line width works in the same way. The line width can not be decreased to be less than 1.

Pressing L2 (figdisp.row) in the bitmap or location windows produces a line plot of the row specified by the cursor position. The plot is labeled "Row Plot, Y=y" or "Row Plot, Average of Rows y to z", depending on the current line width. The horizontal axis is labeled "Column" and the vertical axis is labeled "Data". The values along each axis are periodically labeled.

Two resources, figdisp.row.ascending and figdisp.row.leftToRight, determine how the row plot is produced. If figdisp.row.ascending is True (the default) the plot will always go from lower to higher transformed X values (see the figdisp_cscale() command). If figdisp.row.leftToRight is True (the default) the plot will always go from the left side to the right side of the image. If neither of the resources are true, the plot will go from higher to lower raw (X windows) X values, which is from right to left. If figdisp.row.ascending is True it takes precedence over figdisp.row.leftToRight. Pressing L10 (figdisp.col) in the bitmap or location windows produces a line plot of the column specified by the cursor position. The plot is labeled "Column Plot, X=x" or "Column plot, Average of Columns x to z", depending on the current line width. The horizontal axis is labeled "Row" and the vertical axis is labeled "Data". The values along each axis are periodically labeled.

As with a row plot, two resources, figdisp.col.ascending and figdisp.con.bottomToTop, determine how the column plot is produced. If figdisp.col.ascending is True (the default), the plot will always go from lower to higher transformed Y values. If figdisp.col.bottomToTop is True (the default), the plot will always go from the bottom to the top of the image. If neither of the resources are true, the plot will go from lower to higher raw Y values, which is from top to bottom. If figdisp.col.ascending is True it takes precedence over figdisp.col.bottomToTop.

In addition to row and column plots, figdisp can produce a plot of the data values along an arbitrary line. Press the middle mouse button to select the first end point and release it to select the second end point. Between the selection of the two points a line will appear showing you the line which will be used for the plot. The plot will be labeled "Starting X: x, Starting Y: y, Slope: m" or "Starting X: x, Starting Y: y, Slope: m, Average of z Pixels", depending on the current line width. The slope is calculated such that a line going up from the leftmost end point, regardless of the order in which the points were picked, has a positive slope.

If the absolute value of the slope is greater than 1, the row number is used for the horizontal axis, which is labeled "Row". Otherwise, the column number is used and the horizontal axis is labeled "Column". The vertical axis is always labeled "Data".

The generation of arbitrary line plots is also controlled by two resources, figdisp.line.leftToRight (not to be confused with figdisp.row.leftToRight) and figdisp.line.ascending. If figdisp.line.ascending is True (the default is False), the plot will go from lower to higher values of the coordinate used for the horizontal axis. If figdisp.line.leftToRight is True (the default is False), the leftmost point is used as the starting point and the user specified end points are flipped if necessary. Otherwise, the order in which the user specifies the points is preserved. If figdisp.line.ascending is True it takes precedence over figdisp.line.leftToRight.

2.2.4 The Line Graphics Window

The line graphics window was designed to address the needs of PGPLOT and supplies everything necessary to do so. It uses a default of 16 colors (2 if on a monochrome screen or it can't get 16 colors for some reason), but this can be changed with the lineColors resource.[4] The only other feature provided is a cursor. This is sufficient to support all PGPLOT requirements.

The line graphics window can be resized arbitrarily. The initial size is determined by the figdisp.lg.geometry resource. Since PGPLOT and the above-mentioned line plot utilities automatically scale to use the entire window, the line graphics window should not be resized while a program is accessing it. If this is done, the display server will not crash, but the output will look odd.

The line graphics window title is updated to show the position of the cursor when the cursor is in the line graphics window. If the user has set the lg.crosshair resource to True, the cursor will be two lines, one of which goes vertically from the top to the bottom of the window, the other going horizontally from the left edge to the right edge of the window.

The line plot utilities over-write anything in the line graphics window.

[4]See Section 2.2.10, **Color Allocation**, for a description of how this works.

2.2.5 The "Patch" Window

The "patch" window contains a numerical display of data values around a given point. This point is initially selected by pressing L1 (figdisp.showPatch) in either the bitmap or location windows. When this happens, the patch window appears (or is updated if it was already visible). The chosen point is displayed in reverse video. All numbers are displayed in the font specified by figdisp.font.

The patch window is initially sized so that an 11x11 grid of points is shown, but it can be resized arbitrarily. It can be dismissed by pressing L1 inside the patch window. If any mouse button is pressed inside the patch window, the point under the cursor is moved to the center of the patch window.

2.2.6 The "Location" Window

The "location" window shows how the portion of the image visible in the bitmap window relates to the entire image. This window always displays a scaled copy of the entire image, with a red line around the portion visible in the bitmap window. Many of the bitmap window functions (such as row and column plots) are available in the location window. This includes the ability to click the left mouse button on a point to move that point to the center of the bitmap window.

The initial size of the location window is determined by figdisp.locationgeometry. Unlike most of the other windows, there are restrictions on how much the user can resize the location window. Specifically, the location window can never be larger than 1/2 the image size in either X or Y. Figdisp will also change the size of the location window so that the X and Y dimensions of the image are an integral multiple (or as close as possible) of the X and Y dimensions of the location window. If the resource figdisp.forceSquare is True (the default is False), figdisp will resize the location window so that the scaling factors between the location window and the image are the same for both X and Y. For example, by default the location window can be 1/3 the width of the image and 1/2 the height. If figdisp.forceSquare is True, the location window would be either 1/3 the width and 1/3 the height or 1/2 the width and 1/2 the height.

The location window is dismissed by pressing F8 again.

2.2.7 The Color Map Window

The color map window contains a representation of the entire color map, as affected by the transfer function (right mouse button) and the look-up table inversion (the figdisp.invert key). Every row in the color map window is identical. The leftmost column is the color map entry for a raw data value of 0 and the rightmost column is the color map entry for a raw data value of 65535 (16 bit data) or 255 (8 bit data).

The color map window can be resized arbitrarily. The initial size is determined by figdisp.cmapgeometry.

2.2.8 The "Seeing" Window

The seeing window is produced by pressing L9 (figdisp.showSee) near a star in the bitmap or location windows. Figdisp will then compute the X and Y center and FWHM and display them in the seeing window, using the font specified by figdisp.font. This window can not be resized. It is dismissed by pressing L9 while the cursor is inside the seeing window.

2.2.9 The image statistics window

After changing the dimensions of the image statistics box with the left or right mouse buttons (in the non-default mouse mode), or after pressing "R3" (figdisp.box) a window containing image statistics appears. Those statistics include the mean, standard deviation, minimum, and maximum of the data values in the box, as well as box X and Y ranges, the number of pixels sampled, and the coordinates of the maximum pixel.

The button used to control the mouse mode is controlled by the figdisp.mouseMode resource.

2.2.10 Color Allocation

Figdisp tries to avoid flashing when moving to and from its windows. At the same time, figdisp tries to use as many colors as the user likes. The following steps are used to achieve this goal:

1. Figdisp will try to place both bitmap and line graphics in the default color map, if the default color map is not read-only. If this is possible, no flashing will occur.
2. Figdisp will try to place both bitmap and line graphics in the same private color map. If this is possible, some flashing may occur when moving from a non-figdisp window to a figdisp window (or back), but no flashing will occur when moving between figdisp windows.
3. Figdisp will try to put bitmap graphics in the default color map and line graphics in a private color map. If this is possible, no flashing will occur when moving to or from a bitmap graphics window, but flashing may occur when moving to or from a line graphics window.
4. Figdisp will try to put bitmap graphics in a private color map and line graphics in the default color map. If this is possible, no flashing will occur when moving to or from a line graphics window, but flashing may occur when moving to or from the bitmap graphics window.
5. Figdisp will try to put bitmap and line graphics in two separate private color maps. If this is possible, flashing may occur when moving to or from any figdisp window.
6. If the user has requested more than 16 line graphics colors, figdisp will reduce the number of line graphics colors to 16 and try all previous steps again.
7. Figdisp will try to get a private color map for bitmap graphics. If this is not possible, figdisp will exit. If this is possible, flashing may occur when moving to and from a bitmap graphics window.
8. Figdisp will try to get a read-only color map for line graphics with the user requested number of colors. If this is possible, requests to change the LUT entry for a line graphics color will be ignored. Flashing may occur when moving to and from a line graphics window.
9. If the user requested more than 16 colors, figdisp will try to get a read-only color map with 16 colors.
10. Finally, figdisp will use 2 colors, BlackPixel and WhitePixel, for line graphics.

Whenever figdisp tries to place bitmap graphics in the default color map it uses a user specified maximum (figdisp.bm.maxcolors) and minimum number of colors (figdisp.bm.mincolors). If it is unable to allocate the maximum number of colors, it decreases the number of colors and tries again, until it reaches the minimum number of colors. If it can not get at least the minimum number of colors, it gives up on that step. A similar method is used (with the figdisp.bm.private.maxcolors and figdisp.bm.private.mincolors resources) when figdisp tries to place bitmap graphics in a private color map.

Since figdisp uses the bitmap window's title bar to display data values, if it has to use a private color map it tries to copy the first few (controlled by the figdisp.saveColors resource) colors to the private color map. This means that for the display server to work with a private color map, it must be able to allocate at least figdisp.bm.private.mincolors+figdisp.saveColors color cells in the private color map. If it is not able to do so and the ability to view numeric data values is unimportant, figdisp.saveColors should be reduced (by the user and/or system manager). If the ability to view numeric data values is important and the extra few color map entries are not very important, figdisp.bm.private.mincolors should be reduced. If both are important, you need display which supports more colors. The default value for figdisp.saveColors is 6, which seems to work under OpenWindows 3.0. This will probably have to be changed for other window managers and X display servers.

Finally, the user may want to start up other X programs after he starts figdisp. If those programs must use the default color map and figdisp has used up all available color map entries in the default color map, those programs will not be able to run. To avoid this, change the value of the figdisp.saveColors resource. Figdisp makes sure that, if it uses the default color map, at least figdisp.saveColors color map entries are available for other programs.

Figdisp initializs the bitmap graphics window to contain a 16-bit per pixel test pattern which ranges from 0 to 65535. It sets the bitmap graphics window to contain a linear grey scale with data value 0 corresponding to black and data value 65535 corresponding to white.

The line graphics window is cleared when figdisp begins. The first 16 LUT table entries (or as many of them as possible, if the user requested fewer than 16 colors) are set as specified in Table 2 (possible values range from 0 to 65535). If the line graphics window is using more than 16 colors, the colors above index 15 are initialized to a greyscale with data value 15 corresponding to black (for the purposes of the greyscale - data value 15 is still assigned as specified in table 2) and the maximum data value corresponding to white. If line graphics are attempted on a greyscale screen, all color components are set to the same value, specifically 0.30*red+0.59*green+0.11*blue, where red, green, and blue are the requested values.

2.2.11 Window Managers

Figdisp has two aspects which cause it's behavior to depend on the window manager. The first of these is how figdisp decide which color map to use and has already been discussed. The second is the extensive use of the mouse buttons, which interacts with how the window manager determines input focus.

Most window managers have two methods of determining which window gets input focus: click-to-type, in which the user must click on the window he wants to get input, and follow-pointer, in which the window under the pointer gets input. If the user has specified click-to-type, before he can pan the image, manipulate the LUTs, or any other mouse controlled function, he must click on the figdisp window. If this click is done inside the figdisp window, instead of on the border, figdisp will also get this click and will processes it, which will result in the image being panned (if the left mouse button is used).

Color Index	**Red Component**	**Green Component**	**Blue Component**
0	0	0	0
1	65535	65535	65535
2	65535	0	0
3	0	65535	0
4	0	0	65535
5	0	65535	65535
6	65535	0	65535
7	65535	65535	0
8	65535	32768	0
9	32768	65535	0
10	0	65535	32768
11	0	32768	65535
12	32768	0	65535
13	65535	0	32768
14	21845	21845	21845
15	43690	43690	43690

Table 2: Figdisp Line Graphics Initial Color Map Entries

There are at least two ways around this. The first is to use follow-pointer focus. The way to do this varies from window manager to window manager. For the OpenLook Window Manager, the line

```
OpenWindows.SetInput:     followmouse
```

should be placed into the user's /.Xdefaults file or olwm should be started with the -f or -follow flag. For the Motif window manager, the line

```
Mwm*keyboardFocusPolicy:        pointer
```

accomplishes the same thing.

The second way to prevent unwanted panning in figdisp is by telling the window manager not to pass the select event (clicking on the window) to the program (figdisp). This is accomplished in the Motif window manager by adding the line

```
Mwm*passSelectButton:     false
```

to the user's /.Xdefaults file.

Note that for changes to the .Xdefaults file to take affect, the resource database must be updated and the window manager must be re-started. The easiest way to do this is to exit X windows and start it up again.

2.3 Customizing Figdisp

Figdisp allows the user to customize everything about how it interacts with the user. Table 3 contains the list of command line options and the corresponding X resources that figdisp recognizes. Every resource is prefaced by the name of the program (i.e. a listing of .display means the resource is figdisp.display if the program was started up as figdisp or and pgdisp.display if the program was started up as pgdisp). A detailed discussion of the effects of each of these resources has been given in preceeding sections.

Command Line Option	X resource	Default	Notes
-display disp	.display	none	The display on which the display server should run.
-id #	.id	0	The id number of this copy of figdisp or pgdisp. An arbitrary number of copies of figdisp or pgdisp may be run at the same time, as long as each one's id number is unique.
-geometry WxH[+x+y]		512x512	This flag corresponds to .bm.geometry in figdisp and .lg.geometry in pgdisp
-bmGeometry WxH[+x+y]	.bm.geometry	512x512	The bitmap window geometry.
-lgGeometry WxH[+x+y]	.lg.geometry	512x512	The line graphics window geometry.
-patchGeometry WxH[+x+y]	.patchgeometry	12x12	The geometry of the patch window. The dimensions are the number of rows and columns, including the labels.
-cmapGeometry WxH[+x+y]	.cmapgeometry	256x32	The color map window geometry.
-locationGeometry WxH[+x+y]	.locationgeometry	128x128	The location window (shows the portion of the image visable as compared to the entire image) geometry.
-histogramGeometry WxH	.histgeometry	0x0	See section 2.2.2, Manipulating the Color Maps, for a description of this resource.
-maxColors #	.bm.maxcolors	225	The initial number of colors figdisp will try to use in the default color map for bitmap graphics. Alternate command line argument: -colors
-minColors #	.bm.mincolors	200	The minimum number of colors figdisp will use in the default color map. If this many colors can not be allocated, a private color map will be used.
			continued on next page

continued from previous page

Command Line Option	**X resource**	**Default**	**Notes**
-maxPrivateColors #	.bm.private.maxcolors	65536	The initial number of colors figdisp will try to use in a private color map. This is decreased to the maximum number of colors supported on the chosen display. Alternate command line argument: -privateColors
-minPrivateColors #	.bm.private.mincolors	170	The minimum number of colors figdisp will try to use in a private color map. If this many colors can not be allocated figdisp will give up.
-lineColors	.lineColors	16	The number of colors to use for line graphics.
-visual	.visual	Any	The visual to use. Accepted values include the X11 visuals PseudoColor and GrayScale, as well as Default (only the default visual is allowed) and Any (any visual is allowed) for either pgdisp or figdisp. Pgdisp also allows the X11 visual classes StaticGray, StaticColor, DirectColor, and TrueColor.
-[no]help	.showhelp	True	Whether or not an initial help screen should be displayed.
-font font	.font	fixed	The font to use for all labling. A fixed-width font is recommended but not required. Alternate command line argument: -fn
-psFile outfile	.psFile	direct	The output file for image and window printing.
-printer printer	.printer	PRINTER	The printer to use for image and window printing. Alternate command line argument: -P

continued on next page

continued from previous page

Command Line Option	**X resource**	**Default**	**Notes**
-sleepTime #	.sleepTime	10000	See section 2.4, **Figdisp Design** for a complete description of this resource. Users will usually never need to worry about it.
-[no]forceSquare	.forceSquare	False	If true, the location window will always preserve the aspect ratio of the image (e.g. if the image is twice as wide as it is high, the location window will also be twice as wide as it is high).
-saveColors #	.saveColors	6	The number of colors to try to copy from the default color map to a private color map.
-leaveColors #	.leaveColors	0	The number of colors to leave available in the default color map.
-[no]lgCrosshair	.lg.crosshair	False	If this resource is True, the default cursor will be replaced with a crosshair cursor.
-[no]ascendingCoord	.line.ascending	False	See section 2.2.3, **Line Plots** for a description of this resource. If this resource is true it takes precedence over .line.leftToRight.
-[no]leftToRight	.line.leftToRight	False	See section 2.2.3.
-[no]ascendingX	.row.ascending	True	See section 2.2.3. If this resource is true it takes precedence over .row.leftToRight.
-[no]rowLeftToRight	.row.leftToRight	True	See section 2.2.3.
-[no]ascendingY	.col.ascending	True	See section 2.2.3. If this resource is true it takes precedence over .col.bottomToTop
-[no]bottomToTop	.col.bottomToTop	True	See section 2.2.3.
-[no]plothist	.line.histogram	False	See section 2.2.3.

continued on next page

continued from previous page

Command Line Option	**X resource**	**Default**	**Notes**
-initLUTWrap	.initLUTWrap	1	The initial number of times the LUT should be wrapped (1 means no wrapping).
-zoomin key	.zoomIn	F2	The key used to increase both X and Y zoom factors.
-zoomnorm key	.zoomNorm	F3	The key used to reset both X and Y zoom factors.
-zoomout key	.zoomOut	F4	The key used to decrease both X and Y zoom factors.
-zoomxin key	.x.zoomIn	R7	The key used to increase the X zoom factor.
-zoomxout key	.x.zoomOut	R9	The key used to decrease the X zoom factor.
-zoomyin key	.y.zoomIn	R13	The key used to increase the Y zoom factor.
-zoomyout key	.y.zoomOut	R15	The key used to decrease the Y zoom factor.
-helpkey key	.help	F5	The key used to produce a help screen, including the current key definitions.
-cursor key	.cursor	F6	The key used to toggle the display of the cursor position and data value in the window's title bar.
-recenter key	.recenter	F7	The key used to recenter the image in the window.
-showloc key	.showLoc	F8	The key used to toggle the display of the window showing the main image window's position in the entire window.
-quit key	.quit	F9	The key used to exit figdisp.
-showcmap key	.showCmap	F10	The key used to toggle the display of the window containing the color map in use.

continued on next page

continued from previous page

Command Line Option	**X resource**	**Default**	**Notes**
-showpatch key	.showPatch	L1	The key used to display the window containing numeric pixel values.
-row key	.row	L2	The key used to produce a row plot in the line graphics window.
-imagePrint key	.imagePs	L3	The key used to produce a PostScript output of the entire image.
-windowPrint key	.windowPs	L4	The key used to produce a PostScript output of the portion of the image visible in the main window.
-invert key	.invertCmap	L8	The key used to invert the color map.
-showsee key	.showSee	L9	The key used to produce a window containing the X and Y centroid and FWHM of a star near the cursor.
-column key	.column	L10	The key used to produce a column plot in the line graphics window.
-decreaseSlit key	.decreaseSlit	,	The key used to decrease the slit width used for line plots.
-increaseSlit key	.increaseSlit	.	The key used to increase the slit width used for line plots.
-resetSlit key	.resetSlit	/	The key used to reset the slit width used for line plots.
-inhibit key	.inhibit	L6	The key used to inhibit the interpretation of all other keys. This allows a program which accepts cursor input from figdisp to receive any keyboard value (except this key).
-histogram	.histogram	R1	The key used to cycle between linear (the default), histogram equalization, and limited linear methods of color map allocation.

continued on next page

continued from previous page

Command Line Option	**X resource**	**Default**	**Notes**
-box	.box	R3	The key used to update the image statistics window.
-mouseMode	.mouseMode	R2	The key used to toggle between the default mode for the left and right mouse buttons (where they control panning and color map manipulation) and the mode where they control the image statistics box's corners.
-increaseLUTWrap	.increaseLUTWrap	>	The key used to increase the number of times the LUT is wrapped.
-decreaseLUTWrap	.decreaseLUTWrap	<	The key used to decrease the number of times the LUT is wrapped.
-resetLUTWrap	.resetLUTWrap	?	The key used to reset the number of times the LUT is wrapped to no wrapping.

Table 3: Figdisp Command Line Options and Resources

If you are already familiar with X resources, Table 3, plus the meaning of all the resources, which can be found earlier in this document, is all you need to know. If not, continue reading.

Figdisp looks for its resource values in the following places (precedence is from top to bottom):

- The command line
- The XENVIRONMENT or /.hostname files
- The server resource database (the xrdb command) or the /.Xdefaults file
- The Figdisp or Pgdisp file in XAPPLRESDIR or /usr/lib/X11/app-defaults
- The compile time defaults.

You should note that because of this precedence, the defaults specified by your system manager in the Figdisp or Pgdisp file will take priority over the compile-time defaults, so the listed default values may not agree with those on your system.

2.3.1 Customizing Figdisp with Command-Line Options

The easiest way to experiment with figdisp's options is to use the command line. Doing this is as simple as looking up the command-line option which corresponds to the resource you want to specify, and then using it on the command line. For example, changing the key used to exit figdisp to "q" is as simple as

```
% figdisp -quit q &
```

Some figdisp command-line options do not require an argument. For example, let's say that in addition to changing the quit key, you wanted to disable the initial message figdisp produces referring you to this manual and to the help key. You would do this by the command

```
% figdisp -quit q -nohelp &
```

Now let's say that in addition to all of the above, you wanted to specify that the bitmap graphics window started out 256 pixels wide, 512 pixels tall, 100 pixels from the left edge of the screen, and 200 pixels from the top edge of the screen, the line graphics window to start out 512 pixels wide and 256 pixels high (but didn't care where on the screen it appeared), the font used for labels to be 5x8 (a very small font), PostScript output to go to the file /fdout.ps, the display server to try to allocate a maximum of 128 colors and a minimum of 100 colors in the default color map, and the display server to exit if it has to allocate a private color map. This would take the command

```
% figdisp -quit q -nohelp -bmGeometry 256x512+100+200 \
        -lgGeometry 512x256 -font 5x8 -psFile ~/fdout.ps \
        -colors 128 -minColors 100 -maxPrivateColors 1 \
        -minPrivateColors 2 &
```

Since the minimum number of colors allowable in the private color map is less than the maximum, figdisp will never use a private color map. This technique can also be used to require figdisp to use a private color map.

Note that the options can come in any order and the flags can be abbreviated to the smallest unique string, so the last command could also be

```
% figdisp -minP 2 -maxP 1 -minC 100 -colo 128 -ps ~/fdout.ps \
        -fon 5x8 -lg 512x256 -bm 256x512+100+200 -noh -q q &
```

As was mentioned in Starting Figdisp, if you are using Figaro figdisp has been aliased so that you do not need to specify the "&" to start figdisp in the background.

2.3.2 Customizing Figdisp with Your /.Xdefaults File

As you no doubt realized, typing in a long command line as in the final example can get to be very awkward. This is where the server resource database (which is usually read from the /.Xdefaults file using the command xrdb, usually located in /.xinitrc) comes in handy. The .Xdefaults file has the same format as all the other resource files listed (XENVIRONMENT, /usr/lib/X11/app-defaults/Figdisp, etc), so this section applies to them as well.

Assume that you had decided that the final example in the previous section was how you wanted figdisp to start up. You would then edit your /.Xdefaults file to include the lines:

```
figdisp.quit:                q
figdisp.showhelp:            False
figdisp.bm.geometry:         256x512+100+200
```

```
figdisp.lg.geometry:            512x256
figdisp.font:                   5x8
figdisp.psFile:                 ~/fdout.ps
figdisp.bm.maxcolors:           128
figdisp.bm.mincolors:           100
figdisp.bm.private.maxcolors:   1
figdisp.bm.private.mincolors:   2
```

If your server has a server resource database, you would need to update it by either re-starting X windows or using the xrdb command. Once this was done, figdisp would automatically come up as if you had typed in all those options. You could still override your choices by using a command line option.

2.4 Figdisp Design

This section is for people who need or want to know a little more about how figdisp works. You do not need to read it to use figdisp. However, it may be useful if figdisp acts a little odd - some things about the way figdisp was written will occasionally affect the way things look to the attentive user.

Figdisp has two modes, 8 or 16 bits per pixel. The number of bits per pixel is determined when the image is created, with an argument to the SET_BM_SIZE command. If 16 bit data are written into an 8 bit image the upper 8 bits are used. If 8 bit data are written into a 16 bit image the lower 8 bits are set to 0. Figdisp starts out in 16 bit mode.

Figdisp always considers pixel values to range from 0 to 65535 in 16 bit mode and 0 to 255 in 8 bit mode. Although several pixel values may be mapped to the same color on the screen (depending on the number of colors figdisp allocated and what the user has done with the look-up table entries), all pixel values reported to the user numerically (in the cursor readout and the data values window, for example) retain the full 8 or 16 bits of accuracy. The reported data values can be changed with the figdisp_dscale() routine and the SET_BM_DSCALE command. For example, figdisp can be told to map the data values from 0 to 65535 into the range -509 to 108. Figaro uses this routine to change the displayed data values to match those selected by the user (e.g. the HIGH and LOW parameters in the IMAGE command), so that displayed data values match those in the data file (possibly subject to round-off errors, due to the 8 or 16 bit nature of figdisp).

Since figdisp is an X11 program, all coordinates are relative to the upper left corner, defined to be (0,0). The routine figdisp_cscale() and the command SET_BM_CSCALE can be used to modify the displayed coordinates (cursor output, data values window output, etc) arbitrarily. Figaro uses this image to make coordinates start at the lower left corner and increase up and to the right so that the displayed coordinates match the image's coordinates. The specific coordinate values for the lower left and upper right corners are determined by the command. The IMAGE command sets the lower left corner to (XSTART,YSTART) and the upper right corner to (XEND,YEND).

Figdisp zooms X and Y independently, in powers of two. The only coded-in limit is a zoom factor of 230. The zoom algorithm uses integer bit shifting to be as fast as possible, so odd things happen when it would take more than 32 bits to represent the width shifted by the zoom factor. Since the entire screen is taken up by a single pixel a long time before this limit is reached (unless you have a 100,000 mile wide monitor), this should not normally be a problem.

Figdisp tries not to care if the client program crashes. To do this, it must be able to determine if the client program has gone away without properly closing the communication channel. This is done by periodically looking for the client program. By default, this is done every 10,000 microseconds, and is only done when the connection is inactive. This can be changed using the figdisp.sleepTime resource. This is something the system manager will probably do once (if at all) and set in the application resource file. The user will probably never use this resource. It has a compile-time minimum of scanning every 10,000 microseconds, no matter what the value of figdisp.sleepTime.

3 Programmer's Guide

3.1 Introduction

The figdisp family of programs uses what I believe is the X11 INCR property mechanism for large data transfers, described in section L.2.5 of the X Protocol Reference Manual. Since my understanding of this mechanism is at the very least incomplete, no matter what you need to do, you are encouraged to use the figdisp_opencomm(), figdisp_sendcommand(), figdisp_getresponse(), and figdisp_closecomm() routines, which can be found in the Figaro TVPCKG library. These routines are known to work, no matter how skewed my knowledge of the INCR property mechanism.

If your program only uses line graphics, you are encouraged to use PGPLOT. Not only is PGPLOT very easy to use, it also provides many nice features, including the ability to display output on a wide variety of display devices. Similarly, if you you only bitmap graphics, you are encouraged to use TVPCKG. Both of these systems were designed with a specific object in mind, and therefore might not be suitable for your application. Most obviously, they are written in Fortran, so the calling conventions will seem strange to C programmers.

If you decide not to use TVPCKG for your bitmap graphics applications, you should consider using the figdisp_xxx routines, which are located in the TVPCKG library. Since the bitmap graphics capabilities of figdisp were designed with TVPCKG in mind, every figdisp feature has a corresponding routine.

If you do not want to use the figdisp_xxx routines, or if your program handles both line graphics and bitmap graphics, you must go to the raw data stream and use the figdisp_opencomm(), figdisp_sendcommand(), figdisp_getresponse(), and figdisp_closecomm() routines.

3.2 The figdisp_xxx Routines

In addition to a general description of the function of these routines, several descriptions refer to "sending the XXX" command to the display server. Such a reference should be looked up in section 3.3, **The Raw Command Stream**. Despite many figdisp_xxx arguments being integer (32 bit on Suns), the raw command stream is composed of short (16 bit) quantities. The behavior of any routine which gets a value which can not fit into 16 bits is undefined. All commands which do not require an immediate answer are buffered and may not take place immediately.

3.2.1 figdisp_close - close the channel to the display server

```
void figdisp_close()
```

This routine handles all the detauls of closing the communication link between the client program and the display server. This includes flushing any commands still in the buffer, closing the communication channel, and destroying all X11 items used by the figdisp_xxx routines.

3.2.2 figdisp_closecomm - close the connection to the display server

```
void figdisp_closecomm()
```

This routine ensures that all data from the display server is read (so future clients do not get confused) and closes the communication channel to the display server.

This routine is for programs using the raw command buffer method of communicating with figdisp. Programs which use the normal routines in the figdisp_xxx library (such as figdisp_write, figdisp_dscale, etc.) should use the figdisp_close routine.

3.2.3 figdisp_cls - clear the display server bitmap screen

```
void figdisp_cls()
```

This routine clears the display server bitmap graphics screen.

3.2.4 figdisp_cscale - set the X and Y scaling factors

```
void figdisp_cscale(xmul, xdiv, xoff, ymul, ydiv, yoff)
int xmul, xdiv, xoff, ymul, ydiv, yoff;
```

This routine changes how the cursor position will be displayed when the user has selected cursor output. The displayed X position is calculated by the formula

$$dispx = (rawx + xoff) * (xmul/xdiv),$$

with a similar formula for calculating the Y value. The display server considers the top left corner to be (0, 0).

For example, Figaro prefers to have the bottom left corner be (1, 1). This mapping is accomplished by the call

```
figdisp_cscale (1, 1, 1, -1, -1, -YSIZE);
```

where *YSIZE* is the height of the image in pixels.

3.2.5 figdisp_dscale - set the data scaling factors

```
void figdisp_dscale(dfac, doff)
float *dfac, *doff;
```

This routine changes how a data value will be displayed when the user has selected cursor output. The displayed value is calculated by the formula

$$dispval = rawval * dfac + doff$$

The raw values are between 0 and 255 if an 8-bit image is being displayed and between 0 and 65535 if a 16-bit image is being displayed. Note that the arguments are pointers to floats, not doubles. The code

```
float scale,low;
scale=(high-low)/255.0;
rlow=low;
figdisp_dscale (&scale, &low);
```

would cause the data values to be displayed as ranging from low to high instead of from 0 to 255.

The discrepancy between this routine and the figdisp_cscale() routine is there because it works easier with TVPCKG.

3.2.6 figdisp_flush - flush all output to the display server

```
void figdisp_flush()
```

To obtain maximum possible throughput, many of the figdisp_xxx routines simply buffer the commands to be sent to the display server. The figdisp_flush routine adds a BM_FLUSH command to the command buffer and flushes the command buffer to the display server.

3.2.7 figdisp_getresponse - get a response from the server

```
short *figdisp_getresponse (len)
int *len;
```

This routine is for support of the raw command stream and is not necessary if you are using the figdisp_xxx routines. len is set to the number of shorts in the returned pointer. The memory the returned pointer points to should be freed with XFree() when it is no longer needed. There is currently no provision for timeout, so before calling this routine you should be sure that there is actually going to be a response.

figdisp_getresponse() returns a NULL pointer if it was not able to get data (see below). Otherwise it returns a pointer to a buffer of shorts.

Table 4 contains a list of the error messages figdisp_getresponse() may print, and what they mean.

“Could not get property value!”	figdisp_getresponse() was not able to get the value of the property used to transfer data between the display server and the client. In this case, a value of NULL is returned and len is set to 0.
“OOPS - there was data left!”	For some reason the returned data could not be read all at once. This should never happen. As much data as could be read will still be returned.

Table 4: figdisp_getresponse error messages

3.2.8 figdisp_getsize - get the size of the bitmap graphics image

```
void figdisp_getsize(xmax, ymax, dmax)
int *xmax, *ymax, *dmax;
```

This routines returns some of the significant characteristics of the display server. xmax is set to the maximum allowed horizontal position (values 0 through xmax, inclusive, are valid, 0 being the leftmost column). ymax is set to the maximum allowed vertical position (0 being the topmost row). dmax is set to the maximum allowed data value. Currently, dmax is always 65535, since the display server uses at most 16 bits per pixel.

Do not confuse the size of the bitmap graphics window with the size of the bitmap graphics image.

3.2.9 figdisp_init - initialize the display server

```
void figdisp_init()
```

This routine initializes the display server, not the figdisp_xxx package. This simply means sending the RESET command to the display server and making sure that the image window is visible.

3.2.10 figdisp_line - draw a line into the bitmap window

```
void figdisp_line(x1, y1, x2, y2, val)
int x1, x2, y1, y2, val;
```

A line of the 8 or 16 bit (depending on the current image depth) value *val* is drawn from (*x1*, *y1*) to (*x2*, *y2*).

3.2.11 figdisp_maxbuflen - return the maximum single-transfer buffer size

```
int figdisp_maxbuflen()
```

This routine returns the maximum buffer size (in shorts) which will be transferred all at once. Although the figdisp_sendcommand routine should break any larger requests into smaller pieces, no program currently uses this feature and it may not work. By sizing your buffer to the return value from figdisp_maxbuflen you will avoid any problems.

"Couldn't open display!"	The call to XOpenDisplay failed - either the display server is not running or the user has not set the DISPLAY environment variable properly.
"Can't find the selection atom!"	The program was not able to find the atom used to handle selections. This should never happen unless the display server has not been started.
"Can't find the incremental atom!"	The program was not able to find the atom used to transfer data between the display server and the client program. This should never happen unless the display server has not been started.
"Can't find the locking atom!"	The program was not able to find the atom used to ensure that only on client access the display server at any given time. This should not happen unless the display server has not been started.
"No display server running!"	The program was not able to contact the display server.
"Someone's using the display!"	The display server appears to be in use. This could have been caused by a client program crashing. If so, the condition should fix itself in a little while (under a minute).
"Couldn't own selection atom!"	For some reason the program was not able to get ownership of the selection atom, so data transfers would be impossible.
"Can't own property!"	The program was not able to get ownership of the property the display told it to use for transferring data.

Table 5: figdisp_opencomm() error messages

3.2.12 figdisp_open - open a connection to figdisp and initialize figdisp_xxx

```
int figdisp_open(dev)
int dev;
```

This routine is simply a wrapper around the figdisp_opencomm() routine, except that it uses its own buffer instead of a user-supplied buffer. See the description of figdisp_opencomm() for a description of the return values and errors.

3.2.13 figdisp_opencomm - open a connection to the display server

```
int figdisp\_opencomm(maxbuflen,dev)
int maxbuflen;
int dev;
```

Maxbuflen is the maximum number of shorts in your command buffer. *Dev* is the specific copy of figdisp or pgdisp which should be opened. This routine should only be used by programs which do not use the figdisp_xxx routines such as figdisp_write() and figdisp_line(). If your programs use those routines, you should open the communication channel with the figdisp_open() routine.

figdisp_opencomm() returns 1 on success and 0 on failure. If figdisp_opencomm() fails, it prints an error message to stdout. Table 5 contains a list of the possible error messages.

3.2.14 figdisp_points - write a list of points to the bitmap image

```
void figdisp_points(npts, xs, ys, pts, bppix)
int npts;
int *xs, *ys;
u_char *ptr;
int bppix;
```

This routine write the data values specified in *pts* to the coordinates listed in *xs* and *ys*. If *bppix* is 8, *pts* is interpreted as an array of unsigned characters, as is shown in the declaration. If *bppix* is 16, *pts* is interpreted as an array of unsigned shorts.

For example, the code

```
int xs[2];
int ys[2];
u_short pts[2];

xs[0] = 3;
ys[0] = 5;
pts[0] = 735;
xs[1] = 6;
ys[1] = 2;
pts[1] = 6000;
figdisp_points(2, xs, ys, pts, 16);
```

Would set the image data value at (3, 5) to 735 and the data value at (6, 2) to 6000.

3.2.15 figdisp_readcursor - return cursor position and the button/key pressed

```
void figdisp_readcursor(x, y, val)
int *x, *y, *val;
```

This routine waits for the user to press a key or button (one not assigned to a function such as zoom or pan) and then returns the position in the image in x and y and the value of the key pressed in val. See the description of BM_GET_CURS for details on interpreting val.

3.2.16 figdisp_resize - resize the bitmap image

```
void figdisp_resize(x, y, bppix)
int x, y, bppix;
```

This routine sends a SET_BM_SIZE command to the display server. x and y must be between 0 and 32767 (inclusive). If bppix is not 16, 8 is used. See the description of SET_BM_SIZE for information on what happens if the server is unable to get sufficient memory to store the image.

3.2.17 figdisp_sendcommand - send the given buffer to the display server

```
void figdisp_sendcommand(buf, len)
short *buf;
int len;
```

This command sends the len shorts from buf to the display server. It is only necessary when using the raw command stream method of communicating with the display server. The figdisp_xxx routines use their own buffer and send it when necessary.

3.2.18 figdisp_setcursor - set the position of the bitmap cursor

```
void figdisp_setcursor(x, y)
int x, y;
```

This routine moves the cursor to the specified point in the image. If the position is within the window boundaries, the cursor is moved to the specified point from anywhere on the screen.

3.2.19 figdisp_wlut - modify the color LUTs

```
void figdisp_wlut(cvals, start, nluts, luts)
short *cvals;
int start, nluts, luts;
```

cvals is the array of values (0-65535) to put into the LUT entries. *start* is the first LUT entry to change and *nluts* is the total number to change. If *luts* is 1, 2, or 3, the red, green, or blue (respectively) LUT entries are affected. Otherwise, all LUT entries are set to the same value (*cvals[0]* is the red, green, and blue value for LUT entry *start*).

3.2.20 figdisp_write - write the given pixels to the bitmap image

```
void figdisp_write(x, y, pix, npix, isshort)
int x, y, npix, isshort;
char *pix;
```

This routine writes *npix* pixels on row y, starting from column x. Pixels which would run off the edge of the image are discarded. *isshort* should be set to 0 if the data values are 8-bit. Anything else means the pixels are 16-bit. If *isshort* is zero and the current image is a 16-bit image, the values are used as the most significant byte of the image data and the least significant byte is set to zero. If *isshort* is non-zero and the current image is an 8-bit image, the most significant byte is used as the image data.

Only a single row is written because that is the model which TVPCKG uses. If you desire a more flexible interface, look at the description of the BM_WRITE command. Do not use the raw command stream simply for performance – the figdisp_xxx routines are buffered and the performance penalty from using figdisp_write is minimal.

3.2.21 figdisp_zoompan - set the zoom factor and center of the window

```
void figdisp_zoompan(x, y, zoom)
int x, y, zoom;
```

The point (x, y) is moved to the center of the displayed window and the X and Y zoom factors are set to *zoom* (a power of two).

3.3 The Raw Command Stream

The lowest level at which you can control the figdisp display server is by sending it a stream of commands. This is your only option if you wish to mix line graphics and bitmap graphics. The routines **figdisp_opencomm()**, **figdisp_sendcommand()**, **figdisp_getresponse()**, and **figdisp_closecomm()** (see above) are available to open the channel to the display server, send a command buffer, receive a response, and close the channel to the display server, but you must provide and format the command buffer. This section describes the commands (defined in the file commands.h) which the display server provides. The numbering of commands in commands.h will not be changed (though more commands may be added), but you are still encouraged to use the symbolic definitions in commands.h.

When formatting your command buffer, be sure to get the arguments correct. If you tell the display server to write past the end of the image it will catch it and not do so. However, if you tell it that you are sending 24 pixels and then only send 22 pixels it can not detect this and is likely to start spewing out error messages and may even crash.

All requests must begin on a 16-bit boundary. Any padding necessary to accomplish this will be ignored. Several of the commands described below return responses to the client. Since a response is usually wanted right away, there is no provision for buffering responses. This means that only one response can be generated per command buffer.

Currently there are two different display servers: figdisp and pgdisp. Figdisp understands all commands. Pgdisp (PGplot DISPlay) is for line graphics only, and only understands the commands applicable to line graphics. All other commands will be properly absorbed (without doing anything) and will generate a warning message once per command buffer.

The behavior of an out-of-bounds x or y coordinate or line graphics color index is undefined, but guaranteed not to crash the display server. Coordinates are usually clipped to their minimum or maximum values and the upper bits of color indices are usually ignored. This behavior is not guaranteed unless the protocol definition specifically mentions it.

3.3.1 BM_DEF_SIZE - get default size of bitmap image

This command returns the default size of the bitmap graphics image, which is defined to be the current image size. The values returned are the minimum and maximum valid x and y values. Since the display servers can all resize the image arbitrarily (subject to available memory), this command is only useful when attempting to preserve the contents of the display (as TVPCKG sometimes does).

Request Encoding

# of Bytes	Value	Description
2	22	opcode

Reply Encoding

# of Bytes	Value	Description
2	22	opcode
2	xmin	minimum valid x coordinate
2	xmax	maximum valid x coordinate
2	ymin	minimum valid y coordinate
2	ymax	maximum valid y coordinate

3.3.2 BM_FLUSH - flush bitmap graphics output

This command flushes all bitmap graphics output to the screen. In the current version of figdisp this has no effect, but previous versions required this call to be reasonably efficient. It is conceivable that this command will actually do something in future versions of the display server.

Request Encoding

# of Bytes	Value	Description
2	28	opcode

3.3.3 BM_GET_CURS - return cursor position

This command causes the display server to wait for the user to press a key (not including one used by the display server to do something else) and generate a reply.

If the user pressed a mouse button *isbutton* will be non-zero and *keyval* will be the number of the button which was pressed, from left to right, starting from 0. Otherwise, *isbutton* will be zero and keyval will be the ASCII code of the key which the user pressed.

Request Encoding

# of Bytes	Value	Description
2	19	opcode

Reply Encoding

# of Bytes	Value	Description
2	19	opcode
2	x	X position where the user pressed a key or button
2	y	Y position where the user pressed a key or button
1	isbutton	non-zero if a mouse button was pressed
1	keyval	The code corresponding to the key/button the user pressed

3.3.4 BM_LINE - draw a line on the bitmap graphics image

This command draws a line in the bitmap graphics window of value *val* from (*x1*, *y1*) to (*x2*, *y2*). If the bitmap graphics window is displaying a 16-bit image, the value is multiplied by 256 to get the displayed value.

Request Encoding

# of Bytes	Value	Description
2	29	opcode
2	x1	The X coordinate of the first point.
2	y1	The Y coordinate of the first point.
2	x2	The X coordinate of the second point.
2	y2	The Y coordinate of the second point.
2	val	The value to use for the line (0-255).

3.3.5 BM_MAX_DIM - returm maximum dimensions of the bitmap grapics window

This command returns a compile-time defined maximum dimensions and color capabilities of the bitmap graphics window. This value is fairly irrelevant, since figdisp can handle image up to 65536x65536. However, some programming models require a display device to have a maximum allowable image size.

Request Encoding

# of Bytes	Value	Description
2	21	opcode

Reply Encoding

# of Bytes	Value	Description
2	21	opcode
2	minx	The minimum valid x coordinate. Should always be 0.
2	maxx	The maximum valid x coordinate (not neccesarily related to the current window width).
2	miny	The minimum valid y coordinate. Should always be 0.
2	maxy	The maximum valid y coordinate.
2	mincol	The minimum allowed color index. Should always be 0.
2	maxcol	The maximum allowed color index.

3.3.6 BM_SET_CURS - set the bitmap graphics cursor position

This command sets the bitmap graphics window cursor to the specified point. If the cursor is not near the given position it is warped there from anywhere on the screen.

Request Encoding

# of Bytes	Value	Description
2	18	opcode
2	x	The new X position of the cursor.
2	y	The new Y position of the cursor.

3.3.7 BM_SH_UPDATE - update an area of a shared memory image

The BM_SH_UPDATE command tells figdisp to update its internal data structures for an area of an image shared via the SET_BM_SH_SIZE command. It is the shared memory alternative to BM_WRITE.

If figdisp receives a BM_SH_UPDATE command when the most recent image was resized using the SET_BM_SIZE command (instead of the SET_BM_SH_SIZE), it ignores the BM_SH_UPDATE command.

Request Encoding

# of Bytes	Value	Description
2	33	opcode
2	x	The X coordinate of the upper left corner of the area which has been updated.
2	y	The Y coordinate of the upper left corner of the area which has been updated.
2	width	The width of the area which has been updated.
2	height	The height of the area which has been updated.

3.3.8 BM_WRITE - write data to the bitmap image

This command writes the given 8 or 16 bit data (determined by *bppix*) to the bitmap image. The display server will not write data outside the image, no matter what the values of *x*, *y*, *width*, and *height*. However, it does treat the specified number of bytes as data.

For example, if the current image width is 10, *bppix* is 8, *x* is 8, and *width* is 4, columns 8 and 9 will be updated and the next two bytes of the command stream will be ignored.

Since all commands must begin on a 16 bit boundary, if the area defined by *width* and *height* contains an odd number of pixels and *bppix* is 8, you must include an extra byte at the end of this command (which is ignored).

Request Encoding

# of Bytes	Value	Description
2	24	opcode
2	bppix	The number of bits per pixel (8 or 16).
2	x	The X coordinate of the upper left corner of the affected area.
2	y	The Y coordinate of the upper left corner of the affected area.
2	width	The width of the affected area.
2	height	The height of the affected area.
width*height*C	data	The data to copy into the bitmap image. The pixels are specified left to right along the all rows, the topmost row first. If *bppix* is 8, *C* is 1. Otherwise, *C* is 2.
1	pad	This padding should be here if and only if $width * height$ is odd and *bppix* is 8. If present, the specific value is insignificant.

3.3.9 BM_ZOOMPAN - change image position and zoom factor

This command causes the bitmap graphics window to be centered on the given point in the image and sets the zoom factor to *zfac*, which is interpreted as a power of two. Because of the way zooms are handled, *zfac* should be between -30 and +30. Because the user has complete control over the window center and zoom factor without the client program being notified, it is unwise to use this command unless you want the initial placement and zoom factor of an image to be different from the default (the window showing the central portion of the image and the zoom factor set to 0).

Setting the X or Y coordinate (or both) to -1 leaves them unchanged. Setting either of the zoom factors to 30 leavs them as is.

Request Encoding

# of Bytes	Value	Description
2	27	opcode
2	x	The image X coordinate for the new center of the window.
2	y	The image Y coordinate for the new center of the window.
2	xzoom	The new zoom factor (power of two) for X.
2	yzoom	The new zoom factor (power of two) for Y.

3.3.10 CLR_BM_WIN - clear the bitmap graphics window

This command clears the bitmap graphics window.

Request Encoding

# of Bytes	Value	Description
2	20	opcode

3.3.11 CLR_LG_WIN - clear the line graphics window

This command clears the line graphics window.

Request Encoding

# of Bytes	Value	Description
2	7	opcode

3.3.12 DO_AUTODISP - set range for a modified linear colormap ramp

This command causes the color map entries to be allocated in a linear fashion between values determined by *min* and *max*. *Min* and *max* are percentages. A histogram of the image data values is constructed. The *min* percent of the pixels with the lowest value are assigned to the lowest color map LUT entry, and the $100 - max$ percent of the pixels with the greatest value are assigned to the highest color map LUT entry.

For example, assume the following:

- *min* is 10.0
- *max* is 95.0
- the image being displayed is a 16-bit image (data values from - to 65535)
- 10% of the pixels in the image are less than 65000
- 5% of the pixels are greater than 65100
- figdisp had allocated 101 color map entries (0 through 100)

In this case, pixel values 0 through 65000 would map to LUT entry 0 and pixel values 65100 through 65535 would map to LUT entry 100. Pixel values 65001, 65002, . . . , 65099 would map to LUT entries 1, 2, . . . , 99.

Min and *max* are ASCII-encoded strings, in the same format as the parameters to the SET_BM_DSCALE command.

Request Encoding

# of Bytes	Value	Description
2	35	opcode
len1	min	The percentage of pixels which will be mapped to the lowest LUT entry
len2	max	100 - *max* is the percentage of pixels which will be mapped to the highest LUT entry.
1	pad	This pad value should be present if and only if len1 + len2 is odd. The value is irrelevant.

3.3.13 DO_BOX - recompute statistics inside the statistics box

This command causes the image statistics displayed in the image statistics window to be recomputed. If the image statistics window is not currently visible, it is made visible.

Request Encoding

# of Bytes	Value	Description
2	34	opcode

3.3.14 DO_HISTEQ - set the LUT allocation mode to histogram equalization

This command causes the color LUT entries to be allocated with an equal number of pixels (not pixel values) to each LUT entry. See Section 2.2.2, "Manipulating the Color Maps" for more information.

Request Encoding

# of Bytes	Value	Description
2	36	opcode

3.3.15 DRAW_DOT - draw a dot in the line graphics window

This command draws a dot with a radius of the current line size in the line graphics window at (x, y) in the current color.

Request Encoding

# of Bytes	Value	Description
2	12	opcode
2	x	The x coordinate of the center of the dot.
2	y	The y coordinate of the center of the dot.

3.3.16 DRAW_LINE - draw a line in the line graphics window

This command draws a line in the line graphics window from (*x1*, *y1*) to (*x2*, *y2*) in the current color.

Request Encoding

# of Bytes	Value	Description
2	5	opcode
2	x1	X coordinate of first point
2	y1	Y coordinate of first point
2	x2	X coordinate of second point
2	y2	Y coordinate of second point

3.3.17 DRAW_POLY - draw a poly-line in the line graphics window

This command draws a poly-line in the line graphics window in the current color. *pts* contains *npts* x, y pairs.

Request Encoding

# of Bytes	Value	Description
2	6	opcode
2	npts	The number of points in the poly-line
4*npts	pts	The points (in the same format as DRAW_LINE) to use

3.3.18 FIGDISP_IDLE - wait until the dispay server is idle

This command returns a reply when figdisp is completely idle, except for user input. This primarily means that there is no more input from the client program.

The FIGDISP_IDLE command is useful when a command which causes figdisp to modify the image (such as BM_LINE or FIGDISP_POINTS) is used on a shared memory image. Without FIGDISP_IDLE, there is no guarantee on when a command sent to figdisp is executed. The FIGDISP_IDLE command is the only way a client program can be sure that there are no such commands waiting to be executed, and that it is therefore safe for the client program to modify the shared image.

FIGDISP_IDLE use is necessary, but quite costly - the client program is halted until figdisp has processed all pending commands. FIGDISP_IDLE should be used sparingly.

Request Encoding

# of Bytes	Value	Description
2	37	opcode

Reply Encoding

# of Bytes	Value	Description
2	37	opcode

3.3.19 FIGDISP_POINTS - write a list of points to figdisp

This routine allows a client program to write a list of unrelated points (such as a fit of a spectrum) to figdisp in a single command. The BM_WRITE command is faster, but is only useful when writing a rectangular area of the image. If a client program needs to set a non-rectangular set of points (such as a circle, or just random points), FIGDISP_POINTS is a much better alternative than many BM_WRITE commands to write individual pixels.

Request Encoding

# of Bytes	Value	Description
2	38	opcode
2	bppix	The number of bits per pixel used to interpret the data values (8 or 16).
2	npts	The number of points to write
6*npts	pts	The points to write. This is *npts* triplets of X coordinate, Y coordinate, and data value. If em bppix is 16, data values should be between 0 and 65535. If *bppix* is 8, data values should be between 0 and 255.

3.3.20 FILL_POLY - draw a filled polygon in the line graphics window

This command fills the polygon specified by the given points in the line graphics window with the current color. The polygon is closed automatically if the last point does not coincide with the first point.

Request Encoding

# of Bytes	Value	Description
2	13	opcode
2	npts	The number of points in the polygon.
4*npts	pts	The pairs of x,y coordinates (in the same format as DRAW_LINE) defining the polygon.

3.3.21 FILL_RECT - draw a filled rectangle in the line graphics window

This command fills the rectangle defined by the given points in the line graphics window with the current color. The points do not have to be in any specific order.

Request Encoding

# of Bytes	Value	Description
2	14	opcode
2	x1	The x coordinate of one of the corners of the rectangle.
2	y1	The y coordinate of one of the corners of the rectangle.
2	x2	The x coordinate of the diagonal from the first point.
2	y2	The y coordinate of the diagonal from the first point.

3.3.22 LG_CURS - set the line graphics cursor

This command sets the cursor to the given coordinates. When the user presses a key or a mouse button which isn't used by the display server a reply is generated.

If the position to place the cursor is the same as the last reported cursor position the cursor is not moved. Otherwise, the cursor is warped to the given cursor position from any place on the screen.

If the user pressed a mouse button *isbutton* will be non-zero and *keyval* will be the number of the button which was pressed, from left to right, starting from 0. Otherwise, *isbutton* will be zero and *keyval* will be the ASCII code of the key which the user pressed.

Request Encoding

# of Bytes	Value	Description
2	3	opcode
2	x	X position to place the cursor
2	y	Y position to place the cursor

Reply Encoding

# of Bytes	Value	Description
2	3	opcode
2	x	X position where the user pressed a key or button
2	y	Y position where the user pressed a key or button
1	isbutton	non-zero if a mouse button was pressed
1	keyval	The code corresponding to the key/button the user pressed

3.3.23 LG_DEF_SIZE - get line graphics window default size

This routine returns the default line graphics window size, which is defined to be whatever size the user (or previously run client program) has set it to be.

Request Encoding

# of Bytes	Value	Description
2	10	opcode

Reply Encoding

# of Bytes	Value	Description
2	10	opcode
2	minx	The current minimum valid x coordinate. Should always be 0.
2	maxx	The current maximum valid x coordinate (window width - 1).
2	miny	The current minimum valid y coordinate. Should always be 0.
2	maxy	The current maximum valid y coordinate (window height - 1).

3.3.24 LG_LINE_WID - set line graphics line width

Set the line width (in pixels) for the line graphics window. A width of 0 means use the fastest algorithm available for drawing lines of one pixel width. These types of lines may not meet properly. See the description of XSetLineAttributes() for more information.

Request Encoding

# of Bytes	Value	Description
2	15	opcode
2	width	The new line width.

3.3.25 LG_MAX_DIM - return maximum line graphics dimensions

This command returns the maximum dimensions and color capabilities of the line graphics window. The values returned are fairly irrelevant, since the line graphics window size is only limited by the X server memory. However, some programming models require a display device to have a maximum allowable image size.

Request Encoding

# of Bytes	Value	Description
2	8	opcode

Reply Encoding

# of Bytes	Value	Description
2	8	opcode
2	minx	The minimum valid x coordinate. Should always be 0.
2	maxx	The maximum valid x coordinate (not necessarily related to the current window width).
2	miny	The minimum valid y coordinate. Should always be 0.
2	maxy	The maximum valid y coordinate.
2	mincol	The minimum allowed color index. Should always be 0.
2	maxcol	The maximum allowed color index.

3.3.26 LG_PIXLINE - draw a line of pixels to the line graphics window

This command draws a line of pixels with the specified values to the line graphics window. The mod operator is used to ensure that the specified values are within the allowed range for line graphics colors.

Request Encoding

# of Bytes	Value	Description
2	30	opcode
2	npix	The number of pixels to draw
2	x	The starting X coordinate
2	y	The starting Y coordinate
2*npix	vals	The values for the points to be drawn

3.3.27 LG_SCALE - get information on the line graphics window scale

This command returns information useful for calculating the line graphics window's aspect ratio.

Request Encoding

# of Bytes	Value	Description
2	9	opcode

Reply Encoding

# of Bytes	Value	Description
2	9	opcode
2	mmwidth	The width of the screen (not window) in millimeters.
2	mmheight	The height of the screen in millimeters.
2	pixwidth	The width of the screen in pixels.
2	pixheight	The height of the screen in pixels.

3.3.28 RESET - reset the display server

This command resets the display server's LUTs (both line graphics and bitmap graphics windows, if applicable) and sets the line graphics color and line width to their default values. It does not affect the image or line graphics data.

Request Encoding

# of Bytes	Value	Description
2	0	opcode

3.3.29 SET_BM_CSCALE - set bitmap graphics cursor scaling

This command sets the X and Y scale and offset for the built-in cursor readout. The cursor readout will report a position of

$$(rawx + xoff) * (xmul/xdiv),$$

for the X coordinate where *rawx* is the X position in image coordinates, (0,0) being the top left corner of the image). A similar transformation is applied to the Y coordinates.

Request Encoding

# of Bytes	Value	Description
2	25	opcode
2	xmul	The amount by which the X coordinate should be multiplied.
2	xdiv	The amount by which the X coordinate should be divided.
2	xoff	The amount which should be added to the X coordinate.
2	ymul	The amount by which the Y coordinate should be multiplied.
2	ydiv	The amount by which the Y coordinate should be divided.
2	yoff	The amount which should be added to the Y coordinate.

3.3.30 SET_BM_DSCALE - set bitmap graphics data scaling

This command sets the scaling factor and offset used to transform the raw data values into the values which appear on the screen. Since data transformations are usually arbitrary, while coordinate transformations usually involve simply moving the origin to the lower left corner, a little more effort is required to set the data transformation values.

The following code demonstrates how to format the SET_BM_DSCALE command:

```
float dfac, doff;
short commands[50];
tmpstr1[20];
tmpstr2[20];
char *cptr;
int len1, len2;

sprintf (&tmpstr1[0], "%g", dfac);
sprintf (&tmpstr2[0], "%g", doff);
len1 = strlen (&tmpstr1[0]);
len2 = strlen (&tmpstr2[0]);
commands[0]=SET_BM_DSCALE;
cptr = (char *) &commands[1];
strcpy (cptr, &tmpstr1[0]);
```

```
strcpy (cptr + len1, &tmpstr2[0]);
/* the extra 1 is to pad the buffer if necessary */
commands[(len1 + len2 + 1) / 2] = NEXT_COMMAND;
```

The data value will be displayed as

$$rawval * dfac + doff.$$

Request Encoding

# of Bytes	Value	Description
2	26	opcode
len1	dfac	ASCII-encoded version, in printf "data scaling factor. Len1 is the length of the string, including the trailing NULL.
len2	doff	ASCII-encoded version, in printf "data offset. Len2 is the length of the string, including the trailing NULL.
1	pad	This padding should be here if and only off len1 + len2 is odd. The value is insignificant.

3.3.31 SET_BM_LUT - change the bitmap graphics color look-up tables

This command changes the bitmap graphics window's color LUT entries. *nluts* entries are affected beginning with *start*. *affect* determines how *luts* is interpreted. If *affect* is zero, *luts* contains $3 * nluts$ values, one each for red, green, and blue. If *affect* is 1, the red LUT entries are affected, if 2, the green LUT entries, and if 4, the blue LUTs are affected. If any of the two values are added together, both of the corresponding colors LUT entries will be set to the same value. For example, if *affect* is 3, the red and green LUT entries for *start* will be set to *luts[0]*.

Request Encoding

# of Bytes	Value	Description
2	17	opcode
2	bppix	The number of bits used to number the LUT entries – 8 if there are 256 entires in the color LUT, 16 if there are 65536
2	start	The first LUT entry to be affected.
2	nluts	The number of affected LUT entries.
2	affect	The colors to be affected.
2*nluts or 6*nluts (see above)	luts	The values used to set the LUT entries.

3.3.32 SET_BM_SH_SIZE - set the bitmap graphics image size, using shared memory

This request sets the size of the bitmap graphics window and arranges for the image data to be obtained via shared memory. This is ideal for using figdisp to display images being obtained in real time.

There is currently only one supported "shared memory type" - a SunOS shmget type. To use this type of shared memory, you must first set up a shared memory segment. For example:

```
shmid = shmget (shmkey, 2 * width * height, IPC_CREAT | IPC_EXCL | 0744);
shmptr = shmat (shmid, (char *)0, 0);
```

would be the appropriate setup to create a shared memory segment large enough for a 16-bit per pixel image of size *width* by *height*. The user program would then be able to access the image data via *shmptr*.

Assuming the above code was executed, the SET_BM_SH_SIZE would look something like:

```
combuf[0] = SET_BM_SH_SIZE;
combuf[1] = width;
combuf[2] = height;
combuf[3] = 16;
combuf[4] = 1; /* SunOS shmget shared memory type */
combuf[5] = (shmid >> 16);
combuf[6] = (shmid & 0xFFFF);
```

Figdisp will treat the shared memory area as follows: the first pixel (8 or 16 bits) is the upper left pixel, followed by the pixel to its right, and so on for the first *width* pixels. The row immediately beneath the top row is the next *width* pixels, and so on.

If figdisp is unable to attached to the specified shared memory area for any reason, the *success* field of the reply will be zero. It is then up to the client program to decide whether to abort or continue with the equivalent call to SET_BM_SIZE.

Although it is perfectly legal to use the BM_WRITE command after establishing a shared memory link with the SET_BM_SH_SIZE, it is much more efficient to simply update the memory area and use the BM_SH_UPDATE command.

Request Encoding

# of Bytes	Value	Description
2	32	opcode
2	width	The width of the image
2	height	The height of the image
2	bppix	The number of bits per pixel (8 or 16)
2	shmtype	The shared memory type – currently only type 1 (SunOS) is supported.
2*n	shmid	The information necessary to establish the shared memory link. For *shmtype* 1, this is the shared memory identifier returned by the shmget routine. The value of *n* varies with *shmtype*. For *shmtype* 1, *n* is 2.

Reply Encoding

# of Bytes	Value	Description
2	32	opcode
2	success	0 if figdisp was unable to attach to the shared memory area for any reason, or if the shared memory area was too small for the supplied *width*, *height*, and *bppix*. Non-zero if it was able to attach to the shared memory area.

3.3.33 SET_BM_SIZE - set the bitmap graphics image size

This request sets the current bitmap graphics image size. The display server will attempt to retain as much of the old image as possible and fill any extra area with LUT entry 0. However, memory constraints may prevent this. If the display server is unable to get sufficient memory for the entire image it attempts to decrease the size of the image.

Request Encoding

# of Bytes	Value	Description
2	23	opcode
2	width	The new image width.
2	height	The new image height.
2	bppix	The number of bits per pixel (8 or 16).

3.3.34 SET_LG_COL - set the line graphics color index

This command sets the current line graphics color to *index*. The mod operator is used to make sure that *index* is within the number of colors allocated for line graphics.

Request Encoding

# of Bytes	Value	Description
2	4	opcode
2	index	The color to use for future operations

3.3.35 SET_LG_CSCALE

This command sets the line graphics cursor scaling values applied to the cursor coordinates output in the title bar of the line graphics window. The parameters to this request are ASCII-encoded floats, in the same format as the parameters to SET_BM_DSCALE. The displayed X coordinate is computed according to the formula

$$dispx = (rawx - xoff)/xscale$$

where *dispx* is the displayed X coordinate and *rawx* is the raw X coordinate. A similar transformation is applied to the Y coordinate.

Request Encoding

# of Bytes	Value	Description
2	31	opcode
len1	xoff	The offset to be applied to the line graphics cursor X coordinate.
len2	xscale	The scaling factor to be applied to the line graphics cursor X coordinate.
len3	yoff	The offset to be applied to the line graphics cursor Y coordinate.
len4	yscale	The scaling factor to be applied to the line graphics cursor Y coordinate.
1	pad	This pad byte should be present if and only if len1 + len2 + len3 + len4 is odd. Its value is irrelevant

3.3.36 SET_LG_LUT - modify line graphics color look-up table

This request changes the line graphics window's color LUT entries. *nluts* red, green, and blue entries beginning with *start* are set to *values*.

Request Encoding

# of Bytes	Value	Description
2	2	opcode
2	start	The first affected LUT entry
2	nluts	The number of affected LUT entries
6*nluts	values	The values to assign to the LUT entries

3.3.37 SET_LG_SIZE - set the line graphics window size

This request sets the current line graphics window size. This has an undefined effect on any data currently in the line graphics window.

Request Encoding

# of Bytes	Value	Description
2	11	opcode
2	width	The new window width.
2	height	The new widow height.

3.3.38 SHOW_BM_WIN - reveal/hide the bitmap graphics window

This request causes the bitmap graphics window to appear and come to the top of the window stack or disappear. If *show* is 0 the bitmap graphics window is hidden. Anything else causes the bitmap graphics window to appear.

Request Encoding

# of Bytes	Value	Description
2	16	opcode
2	show	show/hide the bitmap graphics window

3.3.39 SHOW_LG_WIN - reveal/hide the line graphics window

This request causes the line graphics window to appear and come to the top of the window stack or disappear. If *show* is 0 the line graphics window is hidden. Anything else causes the line graphics window to appear.

Request Encoding

# of Bytes	Value	Description
2	1	opcode
2	show	show/hide the line graphics window

3.4 Examples

The hardest part of accessing the display server is opening and closing the device. The following two example programs both perform the following tasks:

- Open a channel to the device
- Reset the display server
- Resize the image to 100x100
- Clear the image
- Draw a box around the edges of the image
- Draw the two diagonals
- Position the cursor in the center of the bitmap image
- Get a cursor input from the user
- Print the cursor position and the key or button the user pressed
- Close the display server.

The first version uses the figdisp_xxx subroutine library and the second uses the raw command stream. These short examples will clearly show you the advantages of using the figdisp_xxx package if you can. Both of these examples are included in the figdisp (and therefore Figaro) distribution.

3.4.1 The figdisp_xxx Version

```
/* The commands.h file is with the rest of the Figdisp distribution. */
#include "commands.h"

main()
{
    int x,y;            /* the returned cursor position */
    int val;            /* the returned cursor value */

    figdisp_open(0);

    /* the next group of commands is really optional - you can place */
    /* whatever you like here.  You may want to always call */
    /* figdisp_init(), but it is not necessary. */

    figdisp_init();
    figdisp_resize(100,100,8);
    figdisp_cls();
    figdisp_line(0,0,99,0,255);
    figdisp_line(99,0,99,99,255);
    figdisp_line(99,99,0,99,255);
    figdisp_line(0,99,0,0,255);
    figdisp_line(0,0,99,99,255);
    figdisp_line(99,0,0,99,255);
    figdisp_setcursor(50,50);
    figdisp_readcursor(&x,&y,&val);

    if (val & 0xFF00) printf("The user pressed mouse button %d",val & 0xFF);
    else printf("The user pressed key '%c'",val);
    printf(" at (%d,%d).\n",x,y);

    /* you should always call this routine to make sure that everything */
    /* you sent was received. */
    figdisp_close();

    exit(0);
}
```

Get the file commands.h as described in the comments. If you are using figdisp with Figaro, compile and link with the command

```
% cc -o exam1 exam1.c -L~figaro/local/libs -ltvpckg -lX11
```

If you are using figdisp without Figaro, compile and line with the command

```
% cc -o exam1 exam1.c -L/figdisp/dir -lfigdisp -lX11
```

Be sure to replace "/figdisp/dir" with the appropriate directory for your system.

In both cases, if the X11 libraries are not in the default library search path, you will have to add a '-L' command line option to tell the linker where to find them. For example,

```
% cc -o exam1 exam1.c -L/figdisp/dir -L/usr/openwin/lib -lfigdisp -lX11
```

Make sure that the display server is running and then run exam1. After pressing a key or mouse button in the bit map graphics window, you should get output like:

```
The user pressed key 'g' at (30,40).
```

Note that the code also shows how to decode a mouse press. Since the bitmap window uses each mouse button for its own purposes, you will never get a mouse press from the bitmap window. However, you could get a mouse press from the line graphics window. The decoding of the return value is the same.

3.4.2 The Raw Command Stream Version

```
/* This routine is an example of the raw command stream interface to figdisp. */

#include "commands.h"
#include <stdio.h>

main()
{
    short *combuf;        /* the command buffer */
    int combuflen;        /* The length of the command buffer */
    short *response;    /* the response from the display server */
    int rlen;        /* the length of the response */
    int next=0;        /* the next buffer place to use */

    short *figdisp_getresponse();

    /* find out the maximum buffer size */
    if ((combuflen=figdisp_maxbuflen()) < 0)
    {
        printf("Unable to get buffer size!\n");
        exit(1);
    }
    if ((combuf=(short *)malloc(combuflen*sizeof(short))) == NULL)
    {
        printf("Unable to get command buffer\n");
        exit(1);
    }

    /* normally we would want to check periodically (before adding each */
    /* command to the command buffer) to see if we were about to exceed */
    /* the buffer length and if so, send the buffer and reset the */
    /* counter.  However, that would just add a lot of garbage to the */
    /* example, so it is not done. */

    /* corresponds to figdisp_init() */
    combuf[next++]=RESET;
    combuf[next++]=SHOW_BM_WIN;
    combuf[next++]=1;

    /* corresponds to figdisp_resize(100,100,8) */
    combuf[next++]=SET_BM_SIZE;
    combuf[next++]=100;
    combuf[next++]=100;
```

```
combuf[next++]=8;

/* corresponds to figdisp_cls() */
combuf[next++]=CLR_BM_WIN;

/* corresponds to figdisp_line(0,0,99,0,255) */
combuf[next++]=BM_LINE;
combuf[next++]=0;
combuf[next++]=0;
combuf[next++]=99;
combuf[next++]=0;
combuf[next++]=255;

/* corresponds to figdisp_line(99,0,99,99,255) */
combuf[next++]=BM_LINE;
combuf[next++]=99;
combuf[next++]=0;
combuf[next++]=99;
combuf[next++]=99;
combuf[next++]=255;

/* corresponds to figdisp_line(99,99,0,99,255) */
combuf[next++]=BM_LINE;
combuf[next++]=99;
combuf[next++]=99;
combuf[next++]=0;
combuf[next++]=99;
combuf[next++]=255;

/* corresponds to figdisp_line(0,99,0,0,255) */
combuf[next++]=BM_LINE;
combuf[next++]=0;
combuf[next++]=99;
combuf[next++]=0;
combuf[next++]=0;
combuf[next++]=255;

/* corresponds to figdisp_line(0,0,99,99,255) */
combuf[next++]=BM_LINE;
combuf[next++]=0;
combuf[next++]=0;
combuf[next++]=99;
combuf[next++]=99;
combuf[next++]=255;
```

```
/* corresponds to figdisp_line(99,0,0,99,255) */
combuf[next++]=BM_LINE;
combuf[next++]=99;
combuf[next++]=0;
combuf[next++]=0;
combuf[next++]=99;
combuf[next++]=255;

/* corresponds to figdisp_setcursor(50,50) */
combuf[next++]=BM_SET_CURS;
combuf[next++]=50;
combuf[next++]=50;

/* this and the figdisp_getresponse() correspond to */
/* figdisp_readcursor() */
combuf[next++]=BM_GET_CURS;

/* open the channel to the display server */
figdisp_opencomm(combuflen*2, 0);

/* send the buffer we formatted above.  Note that we could have */
/* formatted the buffer after the call to figdisp_opencomm() */
figdisp_sendcommand(combuf,next);

/* now get the response from BM_GET_CURS */
if ((response=figdisp_getresponse(&rlen)) == NULL)
    printf("was not able to get cursor value!\n");
else if (response[0] != BM_GET_CURS || rlen != 4)
    printf("We got the wrong response or the wrong length!\n");
else {
    if (response[3] & 0xFF00)
        printf("The user pressed mouse button %d",
            response[3] & 0xFF);
    else printf("The user pressed key '%c'",response[3]);
    printf(" at (%d,%d).\n",response[1],response[2]);
}

/* you may now format more command buffers, send them, and get */
/* responses for as long as you like */

/* close off the communication channel */
figdisp_closecomm();
```

```
    exit(0);
}
```

This code should be placed in a file exam2.c. Get the file commands.h as in the figdisp_xxx example, and compile with a similar command (replacing exam1 with exam2).

Make sure that the display server is running and then run exam2. You should get the same type of output as exam1.