27. GRAPHICS OUTPUT OPERATIONS

Streams are used as the basis for all I/O operations. Files are implemented as streams that can support character printing and reading operations, and file pointer manipulation. An image stream is a type of stream that also provides an interface for graphical operations. All of the operations that can applied to streams can be applied to image streams. For example, an image stream can be passed as the argument to **PRINT**, to print something on an image stream. In addition, special functions are provided to draw lines and curves and perform other graphical operations. Calling these functions on a stream that is not an image stream will generate an error.

27.1 Primitive Graphics Concepts

The Interlisp-D graphics system is based on manipulating bitmaps (rectangular arrays of pixels), positions, regions, and textures. These objects are used by all of the graphics functions.

27.1.1 Positions

A position denotes a point in an X,Y coordinate system. A **POSITION** is an instance of a record with fields **XCOORD** and **YCOORD** and is manipulated with the standard record package facilities. For example, **(create POSITION XCOORD ← 10 YCOORD ← 20)** creates a position representing the point (10,20).

(POSITIONP *X***)** [Function]

Returns *X* if *X* is a position; **NIL** otherwise.

27.1.2 Regions

A Region denotes a rectangular area in a coordinate system. Regions are characterized by the coordinates of their bottom left corner and their width and height. A **REGION** is a record with fields **LEFT**, **BOTTOM**, **WIDTH**, and **HEIGHT**. It can be manipulated with the standard record package facilities. There

are access functions for the **REGION** record that return the **TOP** and **RIGHT** of the region.

The following functions are provided for manipulating regions:

(CREATEREGION *LEFT BOTTOM WIDTH HEIGHT*) [Function]

Returns an instance of the **REGION** record which has *LEFT*, *BOTTOM*, *WIDTH* and *HEIGHT* as respectively its **LEFT**, **BOTTOM**, **WIDTH**, and **HEIGHT** fields.

Example: **(CREATEREGION 10 -20 100 200)** will create a region that denotes a rectangle whose width is 100, whose height is 200, and whose lower left corner is at the position (10,-20).

(REGIONP *X*) [Function]

Returns *X* if *X* is a region, **NIL** otherwise.

(INTERSECTREGIONS $REGION_1$ $REGION_2$ ... $REGION_n$) [NoSpread Function]

Returns a region which is the intersection of a number of regions. Returns **NIL** if the intersection is empty.

(UNIONREGIONS $REGION_1$ $REGION_2$ ... $REGION_n$) [NoSpread Function]

Returns a region which is the union of a number of regions, i.e. the smallest region that contains all of them. Returns **NIL** if there are no regions given.

(REGIONSINTERSECTP *REGION1 REGION2*) [Function]

Returns **T** if *REGION1* intersects *REGION2*. Returns **NIL** if they do not intersect.

(SUBREGIONP *LARGEREGION SMALLREGION*) [Function]

Returns **T** if *SMALLREGION* is a subregion (is equal to or entirely contained in) *LARGEREGION*; otherwise returns **NIL**.

(EXTENDREGION *REGION INCLUDEREGION*) [Function]

Changes (destructively modifies) the region *REGION* so that it includes the region *INCLUDEREGION*. It returns *REGION*.

(MAKEWITHINREGION *REGION LIMITREGION*) [Function]

Changes (destructively modifies) the left and bottom of the region *REGION* so that it is within the region *LIMITREGION*, if possible. If the dimension of *REGION* are larger than *LIMITREGION*, *REGION* is moved to the lower left of *LIMITREGION*. If *LIMITREGION* is **NIL**, the value of the variable **WHOLEDISPLAY** (the screen region) is used. **MAKEWITHINREGION** returns the modified *REGION*.

(INSIDEP *REGION POSORX Y*) [Function]

If *POSORX* and *Y* are numbers, it returns **T** if the point (*POSORX,Y*) is inside of *REGION*. If *POSORX* is a **POSITION**, it returns **T** if *POSORX* is inside of *REGION*. If *REGION* is a **WINDOW**, the window's interior region in window coordinates is used. Otherwise, it returns **NIL**.

27.1.3 Bitmaps

The display primitives manipulate graphical images in the form of bitmaps. A bitmap is a rectangular array of "pixels," each of which is an integer representing the color of one point in the bitmap image. A bitmap is created with a specific number of bits allocated for each pixel. Most bitmaps used for the display screen use one bit per pixel, so that at most two colors can be represented. If a pixel is 0, the corresponding location on the image is white. If a pixel is 1, its location is black. This interpretation can be changed for the display screen with the function **VIDEOCOLOR** (page 30.23). Bitmaps with more than one bit per pixel are used to represent color or grey scale images. Bitmaps use a positive integer coordinate system with the lower left corner pixel at coordinate (0,0). Bitmaps are represented as instances of the datatype **BITMAP**. Bitmaps can be saved on files with the **VARS** file package command (page 17.35).

(BITMAPCREATE *WIDTH HEIGHT BITSPERPIXEL*) [Function]

Creates and returns a new bitmap which is *WIDTH* pixels wide by *HEIGHT* pixels high, with *BITSPERPIXEL* bits per pixel. If *BITSPERPIXEL* is **NIL**, it defaults to 1.

(BITMAPP *X*) [Function]

Returns *X* if *X* is a bitmap, **NIL** otherwise.

(BITMAPWIDTH *BITMAP*) [Function]

Returns the width of *BITMAP* in pixels.

(BITMAPHEIGHT *BITMAP*) [Function]

Returns the height of *BITMAP* in pixels.

(BITSPERPIXEL *BITMAP*) [Function]

Returns the number of bits per pixel of *BITMAP*.

(BITMAPBIT *BITMAP X Y NEWVALUE*) [Function]

If *NEWVALUE* is between 0 and the maximum value for a pixel in *BITMAP*, the pixel (*X,Y*) is changed to *NEWVALUE* and the old

value is returned. If *NEWVALUE* is **NIL**, *BITMAP* is not changed but the value of the pixel is returned. If *NEWVALUE* is anything else, an error is generated. If (*X*,*Y*) is outside the limits of *BITMAP*, 0 is returned and no pixels are changed. *BITMAP* can also be a window or display stream. Note: non-window image streams are "write-only"; the *NEWVALUE* argument must be non-**NIL**.

(BITMAPCOPY *BITMAP***)** [Function]

Returns a new bitmap which is a copy of *BITMAP* (same dimensions, bits per pixel, and contents).

(EXPANDBITMAP *BITMAP WIDTHFACTOR HEIGHTFACTOR***)** [Function]

Returns a new bitmap that is *WIDTHFACTOR* times as wide as *BITMAP* and *HEIGHTFACTOR* times as high. Each pixel of *BITMAP* is copied into a *WIDTHFACTOR* times *HEIGHTFACTOR* block of pixels. If **NIL**, *WIDTHFACTOR* defaults to 4, *HEIGHTFACTOR* to 1.

(SHRINKBITMAP *BITMAP WIDTHFACTOR HEIGHTFACTOR DESTINATIONBITMAP***)** [Function]

Returns a copy of *BITMAP* that has been shrunken by *WIDTHFACTOR* and *HEIGHTFACTOR* in the width and height, respectively. If **NIL**, *WIDTHFACTOR* defaults to 4, *HEIGHTFACTOR* to 1. If *DESTINATIONBITMAP* is not provided, a bitmap that is 1/*WIDTHFACTOR* by 1/*HEIGHTFACTOR* the size of *BITMAP* is created and returned. *WIDTHFACTOR* and *HEIGHTFACTOR* must be positive integers.

(PRINTBITMAP *BITMAP FILE***)** [Function]

Prints the bitmap *BITMAP* on the file *FILE* in a format that can be read back in by **READBITMAP**.

(READBITMAP *FILE***)** [Function]

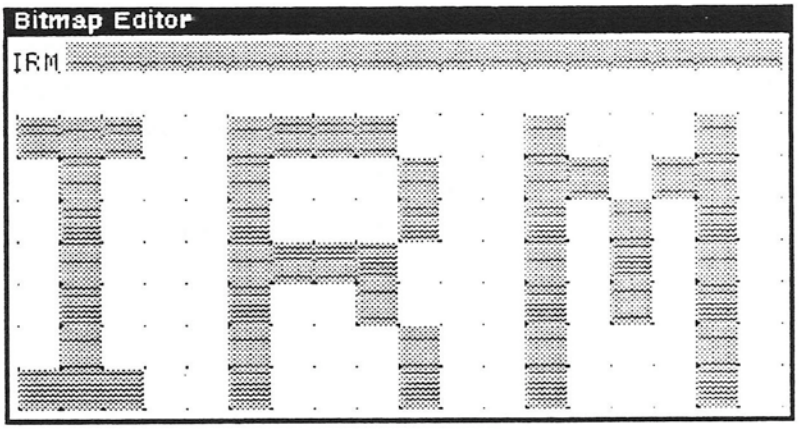
Creates a bitmap by reading an expression (written by **PRINTBITMAP**) from the file *FILE*.

(EDITBM *BMSPEC***)** [Function]

EDITBM provides an easy-to-use interactive editing facility for various types of bitmaps. If *BMSPEC* is a bitmap, it is edited. If *BMSPEC* is an atom whose value is a bitmap, its value is edited. If *BMSPEC* is **NIL**, **EDITBM** asks for dimensions and creates a bitmap. If *BMSPEC* is a region, that portion of the screen bitmap is used. If *BMSPEC* is a window, it is brought to the top and its contents edited.

EDITBM sets up the bitmap being edited in an editing window. The editing window has two major areas: a gridded edit area in

the lower part of the window and a display area in the upper left part. In the edit area, the left button will add points, the middle button will erase points. The right button provides access to the normal window commands to reposition and reshape the window. The actual size bitmap is shown in the display area. For example, the following is a picture of the bitmap editing window editing a eight-high by eighteen-wide bitmap:



If the bitmap is too large to fit in the edit area, only a portion will be editable. This portion can be changed by scrolling both up and down in the left margin and left and right in the bottom margin. Pressing the middle button while in the display area will bring up a menu that allows global placement of the portion of the bitmap being edited. To allow more of the bitmap to be editing at once, the window can be reshaped to make it larger or the **GridSize←** command described below can be used to reduce the size of a bit in the edit area.

The bitmap editing window can be reshaped to provide more or less room for editing. When this happens, the space allocated to the editing area will be changed to fit in the new region.

Whenever the left or middle button is down and the cursor is not in the edit area, the section of the display of the bitmap that is currently in the edit area is complemented. Pressing the left button while not in the edit region will put the lower left 16 x 16 section of the bitmap into the cursor for as long as the left button is held down.

Pressing the middle button while not in either the edit area or the display area (i.e. while in the grey area in the upper right or in the title) will bring up a command menu. There are commands to stop editing, to restore the bitmap to its initial state and to clear the bitmap. Holding the middle button down over a command will result in an explanatory message being printed in the prompt window. The commands are described below:

Paint Puts the current bitmap into a window and call the window **PAINT** command on it. The **PAINT** command implements drawing with various brush sizes and shapes but only on an

actual sized bitmap. The **PAINT** mode is left by pressing the **RIGHT** button and selecting the **QUIT** command from the menu. At this point, you will be given a choice of whether or not the changes you made while in **PAINT** mode should be made to the current bitmap.

ShowAsTile Tesselates the current bitmap in the upper part of the window. This is useful for determining how a bitmap will look if it were made the display background (using the function **CHANGEBACKGROUND**). Note: The tiled display will not automatically change as the bitmap changes; to update it, use the **ShowAsTile** command again.

Grid,On/Off Turns the editing grid display on or off.

GridSize← Allows specification of the size of the editing grid. Another menu will appear giving a choice of several sizes. If one is selected, the editing portion of the bitmap editor will be redrawn using the selected grid size, allowing more or less of the bitmap to be edited without scrolling. The original size is chosen hueristically and is typically about 8. It is particularly useful when editing large bitmaps to set the edit grid size smaller than the original.

Reset Sets all or part of the bitmap to the contents it had when **EDITBM** was called. Another menu will appear giving a choice between resetting the entire bitmap or just the portion that is in the edit area. The second menu also acts as a confirmation, since not selecting one of the choices on this menu results in no action being taken.

Clear Sets all or part of the bitmap to 0. As with the **Reset** command, another menu gives a choice between clearing the entire bitmap or just the portion that is in the edit area.

Cursor← Sets the cursor to the lower left part of the bitmap. This prompts the user to specify the cursor "hot spot" (see page 30.14) by clicking in the lower left corner of the grid.

OK Copies the changed image into the original bitmap, stops the bitmap editor and closes the edit windows. The changes the bitmap editor makes during the interaction occur on a copy of the original bitmap. Unless the bitmap editor is exited via **OK**, no changes are made in the original.

Stop Stops the bitmap editor without making any changes to the original bitmap.

27.1.4 Textures

A Texture denotes a pattern of gray which can be used to (conceptually) tessellate the plane to form an infinite sheet of gray. It is currently either a 4 by 4 pattern or a 16 by N ($N <= 16$)

pattern. Textures are created from bitmaps using the following function:

(CREATETEXTUREFROMBITMAP *BITMAP***)** [Function]

Returns a texture object that will produce the texture of *BITMAP*. If *BITMAP* is too large, its lower left portion is used. If *BITMAP* is too small, it is repeated to fill out the texture.

(TEXTUREP *OBJECT***)** [Function]

Returns *OBJECT* if it is a texture; **NIL** otherwise.

The functions which accept textures (**TEXTUREP**, **BITBLT**, **DSPTEXTURE**, etc.) also accept bitmaps up to 16 bits wide by 16 bits high as textures. When a region is being filled with a bitmap texture, the texture is treated as if it were 16 bits wide (if less, the rest is filled with white space).

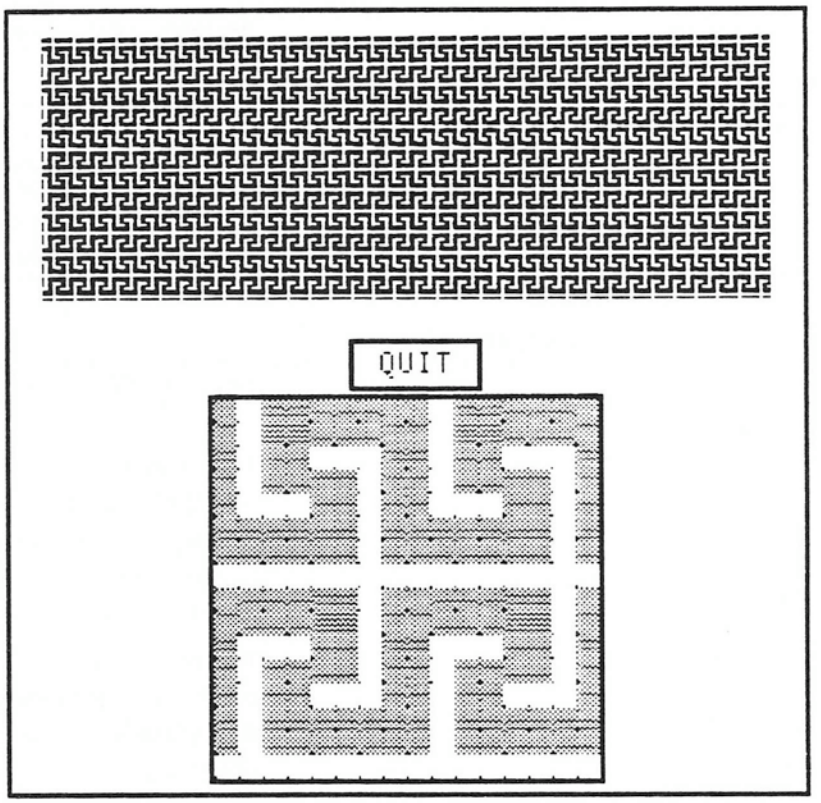
The common textures white and black are available as system constants **WHITESHADE** and **BLACKSHADE**. The global variable **GRAYSHADE** is used by many system facilities as a background gray shade and can be set by the user.

(EDITSHADE *SHADE***)** [Function]

Opens a window that allows the user to edit textures. Textures can be either small (4 by 4) patterns or large (16 by 16). In the edit area, the left button adds bits to the shade and the middle button erases bits from the shade. The top part of the window is painted with the current texture whenever all mouse keys are released. Thus it is possible to directly compare two textures that differ by more than one pixel by holding a mouse key down until all changes are made. When the "quit" button is selected, the texture being edited is returned.

If *SHADE* is a texture object, **EDITSHADE** starts with it. If *SHADE* is **T**, it starts with a large (16 by 16) white texture. Otherwise, it starts with **WHITESHADE**.

The following is a picture of the texture editor, editing a large (16 by 16) pattern:



27.2 Opening Image Streams

An image stream is an output stream which "knows" how to process graphic commands to a graphics output device. Besides accepting the normal character-output functions (**PRINT**, etc.), an image stream can also be passed as an argument to functions to draw curves, to print characters in multiple fonts, and other graphics operations.

Each image stream has an "image stream type," a litatom that specifies the type of graphic output device that the image stream is processing graphics commands for. Currently, the built-in image stream types are **DISPLAY** (for the display screen), **INTERPRESS** (for Interpress format printers), and **PRESS** (for Press format printers). There are also library packages available that define image stream types for the IRIS display, 4045 printer, FX-80 printer, C150 printer, etc.

Image streams to the display (display streams) interpret graphics commands by immediately executing the appropriate operations to cause the desired image to appear on the display screen. Image streams for hardcopy devices such as Interpress printers interpret the graphic commands by saving information in a file, which can later be sent to the printer.

Note: Not all graphics operations can be properly executed for all image stream types. For example, **BITBLT** may not be supported to all printers. This functionality is still being developed, but even in the long run some operations may be beyond the physical or logical capabilities of some devices or image file formats. In these cases, the stream will approximate the specified image as best it can.

(OPENIMAGESTREAM *FILE IMAGETYPE OPTIONS***)** [Function]

Opens and returns an image stream of type *IMAGETYPE* on a destination specified by *FILE*. If *FILE* is a file name on a normal file storage device, the image stream will store graphics commands on the specified file, which can be transmitted to a printer by explicit calls to **LISTFILES** and **SEND.FILE.TO.PRINTER**. If *IMAGETYPE* is **DISPLAY**, then the user is prompted for a window to open. *FILE* in this case will be used as the title of the window.

If *FILE* is a file name on the **LPT** device, this indicates that the graphics commands should be stored in a temporary file, and automatically sent to the printer when the image stream is closed by **CLOSEF**. *FILE* = **NIL** is equivalent to *FILE* = **{LPT}**. File names on the **LPT** device are of the form **{LPT}***PRINTERNAME.TYPE*, where *PRINTERNAME*, *TYPE*, or both may be omitted. *PRINTERNAME* is the name of the particular printer to which the file will be transmitted on closing; it defaults to the first printer on **DEFAULTPRINTINGHOST** that can print *IMAGETYPE* files. The *TYPE* extension supplies the value of *IMAGETYPE* when it is defaulted (see below). **OPENIMAGESTREAM** will generate an error if the specified printer does not accept the kind of file specified by *IMAGETYPE*.

If *IMAGETYPE* is **NIL**, the image type is inferred from the extension field of *FILE* and the **EXTENSIONS** properties in the list **PRINTFILETYPES** (see page 29.6). Thus, the extensions **IP**, **IPR**, and **INTERPRESS** indicate Interpress format, and the extension **PRESS** indicates Press format. If *FILE* is a printer file with no extension (of the form **{LPT}***PRINTERNAME*), then *IMAGETYPE* will be the type that the indicated printer can print. If *FILE* has no extension but is not on the printer device **{LPT}**, then *IMAGETYPE* will default to the type accepted by the first printer on **DEFAULTPRINTINGHOST**.

OPTIONS is a list in property list format, **(***PROP1 VAL1 PROP2 VAL2* **—)**, used to specify certain attributes of the image stream; not all attributes are meaningful or interpreted by all types of image streams. Acceptable properties are:

REGION Value is the region on the page (in stream scale units, 0,0 being the lower-left corner of the page) that text will fill up. It establishes the initial values for **DSPLEFTMARGIN**,

DSPRIGHTMARGIN, **DSPBOTTOMMARGIN** (the point at which carriage returns cause page advancement) and **DSPTOPMARGIN** (where the stream is positioned at the beginning of a new page).

If this property is not given, the value of the variable **DEFAULTPAGEREGION**, is used.

FONTS Value is a list of fonts that are expected to be used in the image stream. Some image streams (e.g. Interpress) are more efficient if the expected fonts are specified in advance, but this is not necessary. The first font in this list will be the initial font of the stream, otherwise the default font for that image stream type will be used.

HEADING Value is the heading to be placed automatically on each page. **NIL** means no heading.

Examples: Suppose that **Tremor:** is an Interpress printer, **Quake** is a Press printer, and **DEFAULTPRINTINGHOST** is **(Tremor: Quake)**:

(OPENIMAGESTREAM) returns an Interpress image stream on printer **Tremor:**.

(OPENIMAGESTREAM NIL 'PRESS) returns a Press stream on **Quake**.

(OPENIMAGESTREAM '{LPT}.INTERPRESS) returns an Interpress stream on **Tremor:**.

(OPENIMAGESTREAM '{CORE}FOO.PRESS) returns a Press stream on the file **{CORE}FOO.PRESS**.

(IMAGESTREAMP *X IMAGETYPE***)** [NoSpread Function]

Returns *X* (possibly coerced to a stream) if it is an output image stream of type *IMAGETYPE* (or of any type if *IMAGETYPE* = **NIL**), otherwise **NIL**.

(IMAGESTREAMTYPE *STREAM***)** [Function]

Returns the image stream type of *STREAM*.

(IMAGESTREAMTYPEP *STREAM TYPE***)** [Function]

Returns **T** if *STREAM* is an image stream of type *TYPE*.

27.3 Accessing Image Stream Fields

The following functions manipulate the fields of an image stream. These functions return the old value (the one being replaced). A value of **NIL** for the new value will return the

current setting without changing it. These functions do not change any of the bits drawn on the image stream; they just affect future operations done on the image stream.

(DSPCLIPPINGREGION *REGION STREAM***)** [Function]

The clipping region is a region that limits the extent of characters printed and lines drawn (in the image stream's coordinate system). Initially set so that no clipping occurs.

Warning: For display streams, the window system maintains the clipping region during window operations. Users should be very careful about changing this field.

(DSPFONT *FONT STREAM***)** [Function]

The font field specifies the font (see page 27.25) used when printing characters to the image stream.

Note: **DSPFONT** determines its new font descriptor from *FONT* by the same coercion rules that **FONTPROP** and **FONTCREATE** use (page 27.26), with one additional possibility: If *FONT* is a list of the form (*PROP*$_1$ *VAL*$_1$ *PROP*$_2$ *VAL*$_2$ ...) where *PROP*$_1$ is acceptable as a font-property to **FONTCOPY** (page 27.28), then the new font is obtained by **(FONTCOPY (DSPFONT NIL** *STREAM***)** *PROP*$_1$ *VAL*$_1$ *PROP*$_2$ *VAL*$_2$ ...). For example, **(DSPFONT '(SIZE 12)** *STREAM***)** would change the font to the 12 point version of the current font, leaving all other font properties the same.

(DSPTOPMARGIN *YPOSITION STREAM***)** [Function]

The top margin is an integer that is the Y position after a new page (in the image stream's coordinate system). This function has no effect on windows.

(DSPBOTTOMMARGIN *YPOSITION STREAM***)** [Function]

The bottom margin is an integer that is the minimum Y position that characters will be printed by **PRIN1** (in the image stream's coordinate system). This function has no effect on windows.

(DSPLEFTMARGIN *XPOSITION STREAM***)** [Function]

The left margin is an integer that is the X position after an end-of-line (in the image stream's coordinate system). Initially the left edge of the clipping region.

(DSPRIGHTMARGIN *XPOSITION STREAM***)** [Function]

The right margin is an integer that is the maximum X position that characters will be printed by **PRIN1** (in the image stream's coordinate system). This is initially the position of the right edge of the window or page.

The line length of a window or image stream (as returned by **LINELENGTH**, page 25.11) is computed by dividing the distance between the left and right margins by the width of an uppercase "A" in the current font. The line length is changed whenever the font, left margin, or right margin are changed or whenever the window is reshaped.

(DSPOPERATION *OPERATION STREAM*) [Function]

The operation is the default **BITBLT** operation (see page 27.15) used when printing or drawing on the image stream. One of **REPLACE**, **PAINT**, **INVERT**, or **ERASE**. Initially **REPLACE**. This is a meaningless operation for most printers which support the model that once dots are deposited on a page they cannot be removed.

(DSPLINEFEED *DELTAY STREAM*) [Function]

The linefeed is an integer that specifies the Y increment for each linefeed, normally negative. Initially minus the height of the initial font.

(DSPSCALE *SCALE STREAM*) [Function]

Returns the scale of the image stream *STREAM*, a number indicating how many units in the streams coordinate system correspond to one printer's point (1/72 of an inch). For example, **DSPSCALE** returns 1 for display streams, and 35.27778 for Interpress and Press streams (the number of micas per printer's point). In order to be device-independent, user graphics programs must either not specify position values absolutely, or must multiply absolute point quantities by the **DSPSCALE** of the destination stream. For example, to set the left margin of the Interpress stream **XX** to one inch, do

(DSPLEFTMARGIN (TIMES 72 (DSPSCALE NIL XX)) XX)

The *SCALE* argument to **DSPSCALE** is currently ignored. In a future release it will enable the scale of the stream to be changed under user control, so that the necessary multiplication will be done internal to the image stream interface. In this case, it would be possible to set the left margin of the Interpress stream **XX** to one inch by doing

(DSPSCALE 1 XX)
(DSPLEFTMARGIN 72 XX)

(DSPSPACEFACTOR *FACTOR STREAM*) [Function]

The space factor is the amount by which to multiply the natural width of all following space characters on *STREAM*; this can be used for the justification of text. The default value is 1. For example, if the natural width of a space in *STREAM*'s current font

is 12 units, and the space factor is set to two, spaces appear 24 units wide. The values returned by **STRINGWIDTH** and **CHARWIDTH** are also affected.

The following two functions only have meaning for image streams that can display color:

(DSPCOLOR *COLOR STREAM***)** [Function]

Sets the default foreground color of *STREAM*. Returns the previous foreground color. If *COLOR* is **NIL**, it returns the current foreground color without changing anything. The default color is white

(DSPBACKCOLOR *COLOR STREAM***)** [Function]

Sets the background color of *STREAM*. Returns the previous background color. If *COLOR* is **NIL**, it returns the current background color without changing anything. The default background color is black.

27.4 Current Position of an Image Stream

Each image stream has a "current position," which is a position (in the image stream's coordinate system) where the next printing operation will start from. The functions which print characters or draw on an image stream update these values appropriately. The following functions are used to explicitly access the current position of an image stream:

(DSPXPOSITION *XPOSITION STREAM***)** [Function]

Returns the X coordinate of the current position of *STREAM*. If *XPOSITION* is non-**NIL**, the X coordinate is set to it (without changing the Y coordinate).

(DSPYPOSITION *YPOSITION STREAM***)** [Function]

Returns the Y coordinate of the current position of *STREAM*. If *YPOSITION* is non-**NIL**, the Y coordinate is set to it (without changing the X coordinate).

(MOVETO *X Y STREAM***)** [Function]

Changes the current position of *STREAM* to the point (*X*,*Y*).

(RELMOVETO *DX DY STREAM***)** [Function]

Changes the current position to the point (*DX*,*DY*) coordinates away from current position of *STREAM*.

(MOVETOUPPERLEFT *STREAM REGION***)** [Function]

Moves the current position to the beginning position of the top line of text. If *REGION* is non-**NIL**, it must be a **REGION** and the X position is changed to the left edge of *REGION* and the Y position changed to the top of *REGION* less the font ascent of *STREAM*. If *REGION* is **NIL**, the X coordinate is changed to the left margin of *STREAM* and the Y coordinate is changed to the top of the clipping region of *STREAM* less the font ascent of *STREAM*.

27.5 Moving Bits Between Bitmaps With BITBLT

BITBLT is the primitive function for moving bits from one bitmap to another, or from a bitmap to an image stream.

(BITBLT *SOURCE SOURCELEFT SOURCEBOTTOM DESTINATION DESTINATIONLEFT DESTINATIONBOTTOM WIDTH HEIGHT SOURCETYPE OPERATION TEXTURE CLIPPINGREGION***)** [Function]

Transfers a rectangular array of bits from *SOURCE* to *DESTINATION*. *SOURCE* can be a bitmap, or a display stream or window, in which case its associated bitmap is used. *DESTINATION* can be a bitmap or an arbitrary image stream.

WIDTH and *HEIGHT* define a pair of rectangles, one in each of the *SOURCE* and *DESTINATION* whose left, bottom corners are at, respectively, (*SOURCELEFT*, *SOURCEBOTTOM*) and (*DESTINATIONLEFT*, *DESTINATIONBOTTOM*). If these rectangles overlap the boundaries of either source or destination they are both reduced in size (without translation) so that they fit within their respective boundaries. If *CLIPPINGREGION* is non-**NIL** it should be a **REGION** and is interpreted as a clipping region within *DESTINATION*; clipping to this region may further reduce the defining rectangles. These (possibly reduced) rectangles define the source and destination rectangles for **BITBLT**.

The mode of transferring bits is defined by *SOURCETYPE* and *OPERATION*. *SOURCETYPE* and *OPERATION* specify whether the source bits should come from *SOURCE* or *TEXTURE*, and how these bits are combined with those of *DESTINATION*. *SOURCETYPE* and *OPERATION* are described further below.

TEXTURE is a texture, as described on page 27.6. **BITBLT** aligns the texture so that the upper-left pixel of the texture coincides with the upper-left pixel of the destination bitmap.

SOURCELEFT, *SOURCEBOTTOM*, *DESTINATIONLEFT*, and *DESTINATIONBOTTOM* default to 0. *WIDTH* and *HEIGHT* default to the width and height of the *SOURCE*. *TEXTURE* defaults to white. *SOURCETYPE* defaults to **INPUT**. *OPERATION* defaults to **REPLACE**. If *CLIPPINGREGION* is not provided, no additional clipping is done. **BITBLT** returns **T** if any bits were moved; **NIL** otherwise.

Note: If *SOURCE* or *DESTINATION* is a window or image stream, the remaining arguments are interpreted as values in the coordinate system of the window or image stream and the operation of **BITBLT** is translated and clipped accordingly. Also, if a window or image stream is used as the destination to **BITBLT**, its clipping region further limits the region involved.

SOURCETYPE specifies whether the source bits should come from the bitmap *SOURCE*, or from the texture *TEXTURE*. *SOURCETYPE* is interpreted as follows:

INPUT The source bits come from *SOURCE*. *TEXTURE* is ignored.

INVERT The source bits are the inverse of the bits from *SOURCE*. *TEXTURE* is ignored.

TEXTURE The source bits come from *TEXTURE*. *SOURCE*, *SOURCELEFT*, and *SOURCEBOTTOM* are ignored.

OPERATION specifies how the source bits (as specified by *SOURCETYPE*) are combined with the bits in *DESTINATION* and stored back into *DESTINATION*. *DESTINATION* is one of the following:

REPLACE All source bits (on or off) replace destination bits.

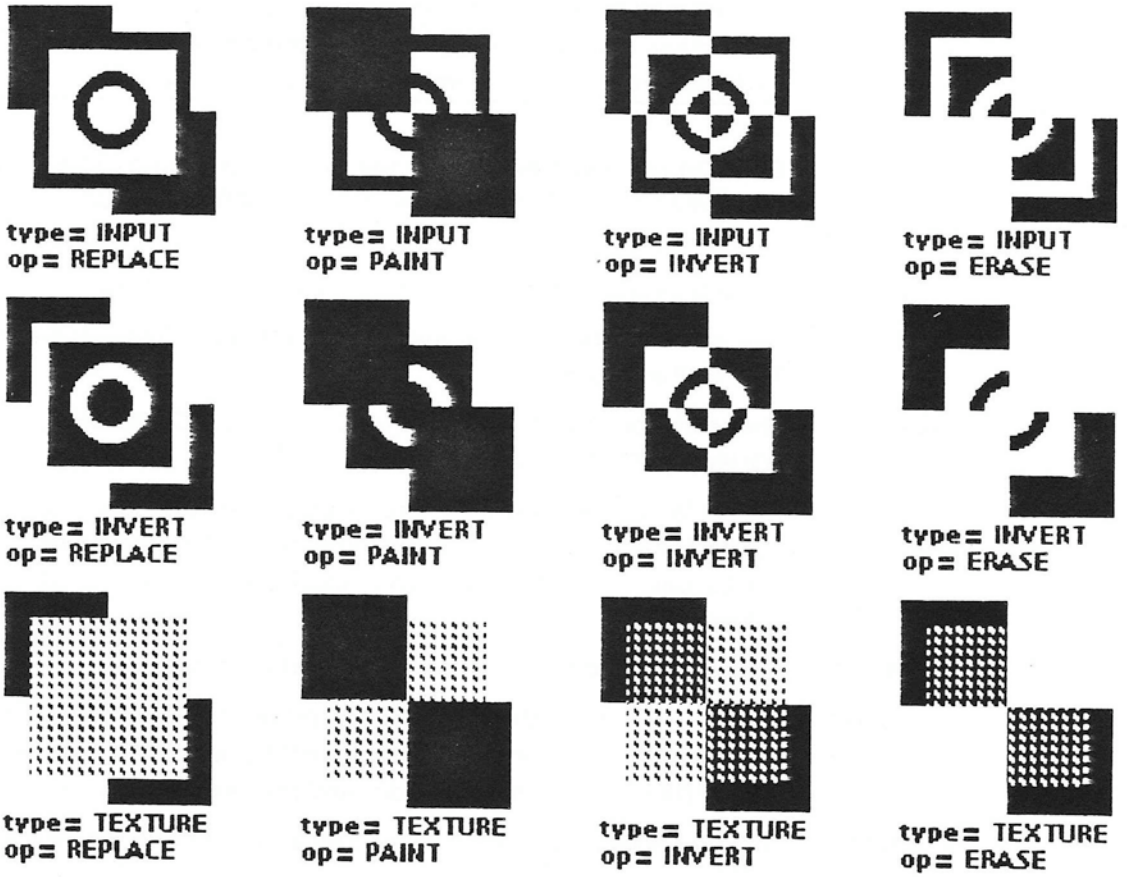
PAINT Any source bits that are on replace the corresponding destination bits. Source bits that are off have no effect. Does a logical OR between the source bits and the destination bits.

INVERT Any source bits that are on invert the corresponding destination bits. Does a logical XOR between the source bits and the destination bits.

ERASE Any source bits that are on erase the corresponding destination bits. Does a logical AND operation between the inverse of the source bits and the destination bits.

Different combinations of *SOURCETYPE* and *OPERATION* can be specified to achieve many different effects. Given the following bitmaps as the values of *SOURCE*, *TEXTURE*, and *DESTINATION*:

BITBLT would produce the results given below for the difference combinations of *SOURCETYPE* and *OPERATION* (assuming *CLIPPINGREGION*, *SOURCELEFT*, etc. are set correctly, of course):



(BLTSHADE *TEXTURE DESTINATION DESTINATIONLEFT DESTINATIONBOTTOM WIDTH HEIGHT OPERATION CLIPPINGREGION*) [Function]

BLTSHADE is the *SOURCETYPE* = **TEXTURE** case of **BITBLT**. It fills the specified region of the destination bitmap *DESTINATION* with the texture *TEXTURE*. *DESTINATION* can be a bitmap or image stream.

(BITMAPIMAGESIZE *BITMAP DIMENSION STREAM*) [Function]

Returns the size that *BITMAP* will be when **BITBLT**ed to *STREAM*, in *STREAM*'s units. *DIMENSION* can be one of **WIDTH**, **HEIGHT**, or **NIL**, in which case the dotted pair (*WIDTH* . *HEIGHT*) will be returned.

27.6 Drawing Lines

Interlisp-D provides several functions for drawing lines and curves on image streams. The line drawing functions are intended for interactive applications where efficiency is important. They do not allow the use of "brush" patterns, like the curve drawing functions, but (for display streams) they support drawing a line in **INVERT** mode, so redrawing the line will erase it. **DRAWCURVE** (page 27.19) can be used to draw lines using a brush.

(DRAWLINE X_1 Y_1 X_2 Y_2 *WIDTH OPERATION STREAM COLOR DASHING***)** [Function]

Draws a straight line from the point (X_1,Y_1) to the point (X_2,Y_2) on the image stream *STREAM*. The position of *STREAM* is set to (X_2,Y_2). If X_1 equals X_2 and Y_1 equals Y_2, a point is drawn at (X_1,Y_1).

WIDTH is the width of the line, in the units of the device. If *WIDTH* is **NIL**, the default is 1.

OPERATION is the **BITBLT** operation (see page 27.15) used to draw the line. If *OPERATION* is **NIL**, the value of **DSPOPERATION** for the image stream is used.

COLOR is a color specification that determines the color used to draw the line for image streams that support color. If *COLOR* is **NIL**, the **DSPCOLOR** of *STREAM* is used.

DASHING is a list of positive integers that determines the dashing characteristics of the line. The line is drawn for the number of points indicated by the first element of the dashing list, is not drawn for the number of points indicated by the second element. The third element indicates how long it will be on again, and so forth. The dashing sequence is repeated from the beginning when the list is exhausted. If *DASHING* is **NIL**, the line is not dashed.

(DRAWBETWEEN $POSITION_1$ $POSITION_2$ *WIDTH OPERATION STREAM COLOR DASHING***)** [Function]

Draws a line from the point $POSITION_1$ to the point $POSITION_2$ onto the destination bitmap of *STREAM*. The position of *STREAM* is set to $POSITION_2$.

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

Draws a line from the current position to the point (X,Y) onto the destination bitmap of *STREAM*. The position of *STREAM* is set to (X,Y).

(RELDRAWTO *DX DY WIDTH OPERATION STREAM COLOR DASHING***)** [Function]

Draws a line from the current position to the point (*DX*,*DY*) coordinates away onto the destination bitmap of *STREAM*. The position of *STREAM* is set to the end of the line. If *DX* and *DY* are both 0, nothing is drawn.

27.7 Drawing Curves

A curve is drawn by placing a brush pattern centered at each point along the curve's trajectory. A brush pattern is defined by its shape, size, and color. The predefined brush shapes are **ROUND**, **SQUARE**, **HORIZONTAL**, **VERTICAL**, and **DIAGONAL**; new brush shapes can be created using the **INSTALLBRUSH** function, described below. A brush size is an integer specifying the width of the brush in the units of the device. The color is a color specification, which is only used if the curve is drawn to an image stream that supports colors.

A brush is specified to the various drawing functions as a list of the form (*SHAPE WIDTH COLOR*), for example **(SQUARE 2)** or **(VERTICAL 4 RED)**. A brush can also be specified as a positive integer, which is interpreted as a **ROUND** brush of that width. If a brush is a litatom, it is assumed to be a function which is called at each point of the curve's trajectory (with three arguments: the X-coordinate of the point, the Y-coordinate, and the image stream), and should do whatever image stream operations are necessary to draw each point. Finally, if a brush is specified as **NIL**, a **(ROUND 1)** brush is used as default.

The appearance of a curve is also determined by its dashing characteristics. Dashing is specified by a list of positive integers. If a curve is dashed, the brush is placed along the trajectory for the number of units indicated by the first element of the dashing list. The brush is *off*, not placed in the bitmap, for a number of units indicated by the second element. The third element indicates how long it will be on again, and so forth. The dashing sequence is repeated from the beginning when the list is exhausted. The units used to measure dashing are the units of the brush. For example, specifying the dashing as **(1 1)** with a brush of **(ROUND 16)** would put the brush on the trajectory, skip 16 points, and put down another brush. A curve is not dashed if the dashing argument to the drawing function is **NIL**.

The curve functions use the image stream's clipping region and operation. Most types of image streams only support the **PAINT** operation when drawing curves. When drawing to a display stream, the curve-drawing functions accept the operation **INVERT** if the brush argument is 1. For brushes larger than 1,

these functions will use the **ERASE** operation instead of **INVERT**. For display streams, the curve-drawing functions treat the **REPLACE** operation the same as **PAINT**.

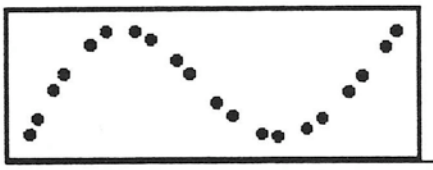
(DRAWCURVE *KNOTS CLOSED BRUSH DASHING STREAM***)** [Function]

Draws a "parametric cubic spline curve" on the image stream *STREAM*. *KNOTS* is a list of positions to which the curve will be fitted. If *CLOSED* is non-**NIL**, the curve will be closed; otherwise it ends at the first and last positions in *KNOTS*. *BRUSH* and *DASHING* are interpreted as described above.

For example,

```
(DRAWCURVE '((10 . 10)(50 . 50)(100 . 10)(150 . 50))
  NIL '(ROUND 5) '(1 1 1 2) XX)
```

would draw a curve like the following on the display stream **XX**:



(DRAWCIRCLE *CENTERX CENTERY RADIUS BRUSH DASHING STREAM***)** [Function]

Draws a circle of radius *RADIUS* about the point (*CENTERX*,*CENTERY*) onto the image stream *STREAM*. *STREAM*'s position is left at (*CENTERX*,*CENTERY*). The other arguments are interpreted as described above.

(DRAWELLIPSE *CENTERX CENTERY SEMIMINORRADIUS SEMIMAJORRADIUS ORIENTATION BRUSH DASHING STREAM***)** [Function]

Draws an ellipse with a minor radius of *SEMIMINORRADIUS* and a major radius of *SEMIMAJORRADIUS* about the point (*CENTERX*,*CENTERY*) onto the image stream *STREAM*. *ORIENTATION* is the angle of the major axis in degrees, positive in the counterclockwise direction. *STREAM*'s position is left at (*CENTERX*,*CENTERY*). The other arguments are interpreted as described above.

New brush shapes can be defined using the following function:

(INSTALLBRUSH *BRUSHNAME BRUSHFN BRUSHARRAY***)** [Function]

Installs a new brush called *BRUSHNAME* with creation-function *BRUSHFN* and optional array *BRUSHARRAY*. *BRUSHFN* should be a function of one argument (a width), which returns a bitmap of the brush for that width. *BRUSHFN* will be called to create new instances of *BRUSHNAME*-type brushes; the sixteen smallest instances will be pre-computed and cached. "Hand-crafted" brushes can be supplied as the *BRUSHARRAY* argument.

Changing an existing brush can be done by calling **INSTALLBRUSH** with new *BRUSHFN* and/or *BRUSHARRAY*.

(DRAWPOINT *X Y BRUSH STREAM OPERATION***)** [Function]

Draws *BRUSH* centered around point (*X*, *Y*) on *STREAM*, using the operation *OPERATION*. *BRUSH* may be a bitmap or a brush.

27.8 Miscellaneous Drawing and Printing Operations

(DSPFILL *REGION TEXTURE OPERATION STREAM***)** [Function]

Fills *REGION* of the image stream *STREAM* (within the clipping region) with the texture *TEXTURE*. If *REGION* is **NIL**, the whole clipping region of *STREAM* is used. If *TEXTURE* or *OPERATION* is **NIL**, the values for *STREAM* are used.

(FILLPOLYGON *POINTS TEXTURE STREAM***)** [Function]

Fills in the polygon outlined by *POINTS* on the image stream *STREAM*, using the texture *TEXTURE*.

POINTS is a list of positions (page 27.1) determining the vertices of a closed polygon. **FILLPOLYGON** fills in this polygon with the texture *TEXTURE*. *POINTS* can also be a list whose elements are lists of positions, in which case each sublist describes a separate polygon to be filled.

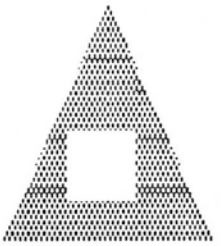
Note: When filling a polygon, there is more than one way of dealing with the situation where two polygon sides intersect, or one polygon is fully inside the other. Currently, **FILLPOLYGON** to a display stream uses the "odd" fill rule, which means that intersecting polygon sides define areas that are filled or not filled somewhat like a checkerboard. For example, **(FILLPOLYGON '((125 . 125)(150 . 200)(175 . 125)(125 . 175)(175 . 175)) GRAYSHADE** *WINDOW***)** would produce a display something like this:

This fill convention also takes into account all polygons in *POINTS*, if it specifies multiple polygons. This can be used to put "holes" in filled polygons. For example,

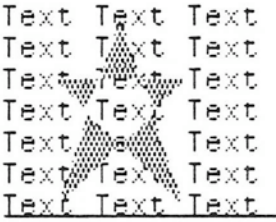
```
(FILLPOLYGON
 '( ((110 . 110)(150 . 200)(190 . 110))
   ((135 . 125)(160 . 125)(160 . 150)(135 . 150)) )
```

GRAYSHADE *WINDOW*)

will put a square hole in a triangular region:



Currently, **FILLPOLYGON** uses the "Replace" **BITBLT** operation (see page 27.15) to fill areas with the texture. However, any areas that are not filled are not changed. If there are "holes" in the filled polygon, this can be used to produce a "window" effect. For example, the following is the display produced by filling the star polygon (above) over a window full of text:



(FILLCIRCLE *CENTERX CENTERY RADIUS TEXTURE STREAM*) [Function]

Fills in a circular area of radius *RADIUS* about the point (*CENTERX*,*CENTERY*) in *STREAM* with *TEXTURE*. *STREAM*'s position is left at (*CENTERX*,*CENTERY*).

(DSPRESET *STREAM*) [Function]

Sets the X coordinate of *STREAM* to its left margin, sets its Y coordinate to the top of the clipping region minus the font ascent. For a display stream, this also fills its destination bitmap with its background texture.

(DSPNEWPAGE *STREAM*) [Function]

Starts a new page. The X coordinate is set to the left margin, and the Y coordinate is set to the top margin plus the linefeed.

(CENTERPRINTINREGION *EXP REGION STREAM*) [Function]

Prints *EXP* so that is it centered within *REGION* of the *STREAM*. If *REGION* is **NIL**, *EXP* will be centered in the clipping region of *STREAM*.

27.9 Drawing and Shading Grids

A grid is a partitioning of an arbitrary coordinate system (hereafter referred to as the "source system") into rectangles. This section describes functions that operate on grids. It includes functions to draw the outline of a grid, to translate between positions in a source system and grid coordinates (the coordinates of the rectangle which contains a given position), and to shade grid rectangles. A grid is defined by its "unit grid," a region (called a grid specification) which is the origin rectangle of the grid in terms of the source system. Its **LEFT** field is interpreted as the X-coordinate of the left edge of the origin rectangle, its **BOTTOM** field is the Y-coordinate of the bottom edge of the origin rectangle, its **WIDTH** is the width of the grid rectangles, and its **HEIGHT** is the height of the grid rectangles.

(GRID *GRIDSPEC WIDTH HEIGHT BORDER STREAM GRIDSHADE***)** [Function]

Outlines the grid defined by *GRIDSPEC* which is *WIDTH* rectangles wide and *HEIGHT* rectangles high on *STREAM*. Each box in the grid has a border within it that is *BORDER* points on each side; so the resulting lines in the grid are 2**BORDER* thick. If *BORDER* is the atom **POINT**, instead of a border the lower left point of each grid rectangle will be turned on. If *GRIDSHADE* is non-**NIL**, it should be a texture and the border lines will be drawn using that texture.

(SHADEGRIDBOX *X Y SHADE OPERATION GRIDSPEC GRIDBORDER STREAM***)** [Function]

Shades the grid rectangle (*X*,*Y*) of *GRIDSPEC* with texture *SHADE* using *OPERATION* on *STREAM*. *GRIDBORDER* is interpreted the same as for **GRID**.

The following two functions map from the X,Y coordinates of the source system into the grid X,Y coordinates:

(GRIDXCOORD *XCOORD GRIDSPEC***)** [Function]

Returns the grid X-coordinate (in the grid specified by *GRIDSPEC*) that contains the source system X-coordinate *XCOORD*.

(GRIDYCOORD *YCOORD GRIDSPEC***)** [Function]

Returns the grid Y-coordinate (in the grid specified by *GRIDSPEC*) that contains the source system Y-coordinate *YCOORD*.

The following two functions map from the grid X,Y coordinates into the X,Y coordinates of the source system:

(LEFTOFGRIDCOORD *GRIDX GRIDSPEC***)** [Function]

Returns the source system X-coordinate of the left edge of a grid rectangle at grid X-coordinate *GRIDX* (in the grid specified by *GRIDSPEC*).

(BOTTOMOFGRIDCOORD *GRIDY GRIDSPEC***)** [Function]

Returns the source system Y-coordinate of the bottom edge of a grid rectangle at grid Y-coordinate *GRIDY* (in the grid specified by *GRIDSPEC*).

27.10 Display Streams

Display streams (image streams of type **DISPLAY**) are used to control graphic output operations to a bitmap, known as the "destination" bitmap of the display stream. For each window on the screen, there is an associated display stream which controls graphics operations to a specific part of the screen bitmap. Any of the functions that take a display stream will also take a window, and use the associated display stream. Display streams can also have a destination bitmap that is not connected to any window or display device.

(DSPCREATE *DESTINATION***)** [Function]

Creates and returns a display stream. If *DESTINATION* is specified, it is used as the destination bitmap, otherwise the screen bitmap is used.

(DSPDESTINATION *DESTINATION DISPLAYSTREAM***)** [Function]

Returns the current destination bitmap for *DISPLAYSTREAM*, setting it to *DESTINATION* if non-**NIL**. *DESTINATION* can be either the screen bitmap, or an auxilliary bitmap in order to construct figures, possibly save them, and then display them in a single operation.

Warning: The window system maintains the destination of a window's display stream. Users should be very careful about changing this field.

(DSPXOFFSET *XOFFSET DISPLAYSTREAM***)** [Function]

(DSPYOFFSET *YOFFSET DISPLAYSTREAM***)** [Function]

Each display stream has its own coordinate system, separate from the coordinate system of its destination bitmap. Having the coordinate system local to the display stream allows objects to be displayed at different places by translating the display stream's

coordinate system relative to its destination bitmap. This local coordinate system is defined by the X offset and Y offset.

DSPXOFFSET returns the current X offset for *DISPLAYSTREAM*, the X origin of the display stream's coordinate system in the destination bitmap's coordinate system. It is set to *XOFFSET* if non-**NIL**.

DSPYOFFSET returns the current Y offset for *DISPLAYSTREAM*, the Y origin of the display stream's coordinate system in the destination bitmap's coordinate system. It is set to *YOFFSET* if non-**NIL**.

The X offset and Y offset for a display stream are both initially 0 (no X or Y-coordinate translation).

Warning: The window system maintains the X and Y offset of a window's display stream. Users should be very careful about changing these fields.

(DSPTEXTURE *TEXTURE DISPLAYSTREAM***)** [Function]

Returns the current texture used as the background pattern for *DISPLAYSTREAM*. It is set to *TEXTURE* if non-**NIL**. Initially the value of **WHITESHADE**.

(DSPSOURCETYPE *SOURCETYPE DISPLAYSTREAM***)** [Function]

Returns the current **BITBLT** sourcetype used when printing characters to the display stream (see page 27.15). It is set to **SOURCETYPE**, if non-**NIL**. Must be either **INPUT** or **INVERT**. Initially **INPUT**.

(DSPSCROLL *SWITCHSETTING DISPLAYSTREAM***)** [Function]

Returns the current value of the "scroll flag," a flag that determines the scrolling behavior of the display stream; either **ON** or **OFF**. If **ON**, the bits in the display streams's destination bitmap are moved after any linefeed that moves the current position out of the destination bitmap. Any bits moved out of the current clipping region are lost. Does not adjust the X offset, Y offset, or clipping region of the display stream. Initially **OFF**.

Sets the scroll flag to *SWITCHSETTING*, if non-**NIL**.

Note: The word "scrolling" also describes the use of "scroll bars" on the left and bottom of a window to move an object displayed in a window. This feature is described on page 28.23.

Each window has an associated display stream. To get the window of a particular display stream, use **WFROMDS**:

(WFROMDS *DISPLAYSTREAM DONTCREATE*) [Function]

Returns the window associated with *DISPLAYSTREAM*, creating a window if one does not exist (and *DONTCREATE* is **NIL**). Returns **NIL** if the destination of *DISPLAYSTREAM* is not a screen bitmap that supports a window system.

If *DONTCREATE* is non-**NIL**, **WFROMDS** will never create a window, and returns **NIL** if *DISPLAYSTREAM* does not have an associated window.

TTYDISPLAYSTREAM calls **WFROMDS** with *DONTCREATE* = **T**, so it will not create a window unnecessarily. Also, if **WFROMDS** does create a window, it calls **CREATEW** with *NOOPENFLG* = **T**.

(DSPBACKUP *WIDTH DISPLAYSTREAM*) [Function]

Backs up *DISPLAYSTREAM* over a character which is *WIDTH* screen points wide. **DSPBACKUP** fills the backed over area with the display stream's background texture and decreases the X position by *WIDTH*. If this would put the X position less than *DISPLAYSTREAM*'s left margin, its operation is stopped at the left margin. It returns **T** if any bits were written, **NIL** otherwise.

27.12 Fonts

A font is the collection of images that are printed or displayed when characters are output to a graphic output device. Some simple displays and printers can only print characters using one font. Bitmap displays and graphic printers can print characters using a large number of fonts.

Fonts are identified by a distinctive style or family (such as Modern or Classic), a size (such as 10 points), and a face (such as bold or italic). Fonts also have a rotation that indicates the orientation of characters on the screen or page. A normal horizontal font (also called a portrait font) has a rotation of 0; the rotation of a vertical (landscape) font is 90 degrees. While any combination can be specified, in practice the user will find that only certain combinations of families, sizes, faces, and rotations are available for any graphic output device.

To specify a font to the functions described below, a **FAMILY** is represented by a literal atom, a **SIZE** by a positive integer, and a **FACE** by a three-element list of the form **(WEIGHT SLOPE EXPANSION)**. **WEIGHT**, which indicates the thickness of the characters, can be **BOLD**, **MEDIUM**, or **LIGHT**; **SLOPE** can be **ITALIC** or **REGULAR**; and **EXPANSION** can be **REGULAR**, **COMPRESSED**, or **EXPANDED**, indicating how spread out the characters are. For convenience, faces may also be specified by

three-character atoms, where each character is the first letter of the corresponding field. Thus, **MRR** is a synonym for **(MEDIUM REGULAR REGULAR)**. In addition, certain common face combinations may be indicated by special literal atoms:

STANDARD = (MEDIUM REGULAR REGULAR) = MRR

ITALIC = (MEDIUM ITALIC REGULAR) = MIR

BOLD = (BOLD REGULAR REGULAR) = BRR

BOLDITALIC = (BOLD ITALIC REGULAR) = BIR

Interlisp represents all the information related to a font in an object called a font descriptor. Font descriptors contain the family, size, etc. properties used to represent the font. In addition, for each character in the font, the font descriptor contains width information for the character and (for display fonts) a bitmap containing the picture of the character.

The font functions can take fonts specified in a variety of different ways. **DSPFONT**, **FONTCREATE**, **FONTCOPY**, etc. can be applied to font descriptors, "font lists" such as **'(MODERN 10)**, image streams (coerced to its current font), or windows (coerced to the current font of its display stream). The printout command "**.FONT**" (page 25.27) will also accept fonts specified in any of these forms.

(FONTCREATE *FAMILY SIZE FACE ROTATION DEVICE NOERRORFLG CHARSET***)** [Function]

Returns a font descriptor for the specified font. *FAMILY* is a litatom specifying the font family. *SIZE* is an integer indicating the size of the font in points. *FACE* specifies the face characteristics in one of the formats listed above; if *FACE* is **NIL**, **STANDARD** is used. *ROTATION*, which specifies the orientation of the font, is 0 (or **NIL**) for a portrait font and 90 for a landscape font. *DEVICE* indicates the output device for the font, and can be any image stream type (page 27.8), such as **DISPLAY**, **INTERPRESS**, etc. *DEVICE* may also be an image stream, in which case the type of the stream determines the font device. *DEVICE* defaults to **DISPLAY**.

The *FAMILY* argument to **FONTCREATE** may also be a list, in which case it is interpreted as a font-specification quintuple, a list of the form (*FAMILY SIZE FACE ROTATION DEVICE*). Thus, **(FONTCREATE '(GACHA 10 BOLD))** is equivalent to **(FONTCREATE 'GACHA 10 'BOLD)**. *FAMILY* may also be a font descriptor, in which case that descriptor is simply returned.

If a font descriptor has already been created for the specified font, **FONTCREATE** simply returns it. If it has not been created, **FONTCREATE** has to read the font information from a font file that contains the information for that font. The name of an appropriate font file, and the algorithm for searching depends on the device that the font is for, and is described in more detail

below. If an appropriate font file is found, it is read into a font descriptor. If no file is found, for **DISPLAY** fonts **FONTCREATE** looks for fonts with less face information and fakes the remaining faces (such as by doubling the bit pattern of each character or slanting it). For hardcopy printer fonts, there is no acceptable faking algorithm.

If no acceptable font is found, the action of **FONTCREATE** is determined by *NOERRORFLG*. If *NOERRORFLG* is **NIL**, it generates a **FONT NOT FOUND** error with the offending font specification; otherwise, **FONTCREATE** returns **NIL**.

CHARSET is the character set which will be read to create the font. Defaults to 0. For more information on character sets, see NS Characters, page 2.12.

(FONTP *X***)** [Function]

Returns *X* if *X* is a font descriptor; **NIL** otherwise.

(FONTPROP *FONT PROP***)** [Function]

Returns the value of the *PROP* property of font *FONT*. The following font properties are recognized:

FAMILY The style of the font, represented as a literal atom, such as **CLASSIC** or **MODERN**.

SIZE A positive integer giving the size of the font, in printer's points (1/72 of an inch).

WEIGHT The thickness of the characters; one of **BOLD**, **MEDIUM**, or **LIGHT**.

SLOPE The "slope" of the characters in the font; one of **ITALIC** or **REGULAR**.

EXPANSION The extent to which the characters in the font are spread out; one of **REGULAR**, **COMPRESSED**, or **EXPANDED**. Most available fonts have **EXPANSION = REGULAR**.

FACE A three-element list of the form (*WEIGHT SLOPE EXPANSION*), giving all of the typeface parameters.

ROTATION An integer that gives the orientation of the font characters on the screen or page, in degrees. A normal horizontal font (also called a portrait font) has a rotation of 0; the rotation of a vertical (landscape) font is 90.

DEVICE The device that the font can be printed on; one of **DISPLAY**, **INTERPRESS**, etc.

ASCENT An integer giving the maximum height of any character in the font from its base line (the printing position). The top line will be at *BASELINE* + *ASCENT*-1.

DESCENT An integer giving the maximum extent of any character below the base line, such as the lower part of a "p". The bottom line of a character will be at *BASELINE-DESCENT*.

HEIGHT Equal to **ASCENT** + **DESCENT**.

SPEC The (*FAMILY SIZE FACE ROTATION DEVICE*) quintuple by which the font is known to Lisp.

DEVICESPEC The (*FAMILY SIZE FACE ROTATION DEVICE*) quintuple that identifies what will be used to represent the font on the display or printer. It will differ from the **SPEC** property only if an implicit coercion is done to approximate the specified font with one that actually exists on the device.

SCALE The units per printer's point (1/72 of an inch) in which the font is measured. For example, this is 35.27778 (the number of micas per printer's point) for Interpress fonts, which are measured in terms of micas.

(FONTCOPY *OLDFONT PROP*$_1$ *VAL*$_1$ *PROP*$_2$ *VAL*$_2$ ...**)** [NoSpread Function]

Returns a font descriptor that is a copy of the font *OLDFONT*, but which differs from *OLDFONT* in that *OLDFONT*'s properties are replaced by the specified properties and values. Thus, **(FONTCOPY** *FONT* **'WEIGHT 'BOLD 'DEVICE 'INTERPRESS)** will return a bold Interpress font with all other properties the same as those of *FONT*. **FONTCOPY** accepts the properties **FAMILY**, **SIZE**, **WEIGHT**, **SLOPE**, **EXPANSION**, **FACE**, **ROTATION**, and **DEVICE**. If the first property is a list, it is taken to be the *PROP*$_1$ *VAL*$_1$ *PROP*$_2$ *VAL*$_2$ ... sequence. Thus, **(FONTCOPY** *FONT* **'(WEIGHT BOLD DEVICE INTERPRESS))** is equivalent to the example above.

If the property **NOERROR** is specified with value non-**NIL**, **FONTCOPY** will return **NIL** rather than causing an error if the specified font cannot be created.

(FONTSAVAILABLE *FAMILY SIZE FACE ROTATION DEVICE CHECKFILESTOO?***)** [Function]

Returns a list of available fonts that match the given specification. *FAMILY*, *SIZE*, *FACE*, *ROTATION*, and *DEVICE* are the same as for **FONTCREATE**. Additionally, any of them can be the atom *, in which case all values of that field are matched.

If *CHECKFILESTOO?* is **NIL**, only fonts already loaded into virtual memory will be considered. If *CHECKFILESTOO?* is non-**NIL**, the font directories for the specified device will be searched. When checking font files, the *ROTATION* is ignored.

Note: The search is conditional on the status of the server which holds the font. Thus a file server crash may prevent **FONTCREATE** from finding a file that an earlier **FONTSAVAILABLE** returned.

Each element of the list returned will be of the form (*FAMILY SIZE FACE ROTATION DEVICE*).

Examples:

(FONTSAVAILABLE 'MODERN 10 'MRR 0 'DISPLAY)

will return **((MODERN 10 (MEDIUM REGULAR REGULAR) 0 DISPLAY))** if the regular Modern 10 font for the display is in virtual memory; **NIL** otherwise.

(FONTSAVAILABLE '* 14 '* '* 'INTERPRESS T)

will return a list of all the size 14 Interpress fonts, whether they are in virtual memory or in font files.

Warning: One must be careful when using the function **FONTSAVAILABLE** to determine what Press font files are available. For Press font families/faces, the font widths for different sizes are consistently scaled versions of the smallest font in the family/face. Therefore, instead of storing data about all of the sizes in the **FONTS.WIDTHS** file, only the widths for the font of *SIZE* = 1 are stored, and the other widths are calculated by scaling these widths up. This is signified in the **FONTS.WIDTHS** file by a font with *SIZE* = 0. Therefore, if **FONTSAVAILABLE** is called with *CHECKFILESTOO?* = **T**, and it finds such a "relative" font, it returns a font spec list with size of 0. For example,

←(FONTSAVAILABLE 'GACHA '* '* 0 'PRESS T)
((GACHA 0 (BOLD ITALIC REGULAR) 0 PRESS)
(GACHA 0 (BOLD REGULAR REGULAR) 0 PRESS)
(GACHA 0 (MEDIUM ITALIC REGULAR) 0 PRESS)
(GACHA 0 (MEDIUM REGULAR REGULAR) 0 PRESS))

This indicates that Press files can be created with **GACHA** files of any size with faces **BIR**, **BRR**, **MIR**, and **MRR**. Of course, this doesn't guarantee that these fonts are available in all sizes on your printer.

(SETFONTDESCRIPTOR *FAMILY SIZE FACE ROTATION DEVICE FONT***)** [Function]

Indicates to the system that *FONT* is the font that should be associated with the *FAMILY SIZE FACE ROTATION DEVICE* characteristics. If *FONT* is **NIL**, the font associated with these characteristics is cleared and will be recreated the next time it is needed. As with **FONTPROP** and **FONTCOPY**, *FONT* is coerced to a font descriptor if it is not one already.

This functions is useful when it is desirable to simulate an unavailable font or to use a font with characteristics different from the interpretations provided by the system.

(DEFAULTFONT *DEVICE FONT* —**)** [Function]

Returns the font that would be used as the default (if **NIL** were specified as a font argument) for image stream type *DEVICE*. If *FONT* is a font descriptor, it is set to be the default font for *DEVICE*.

(CHARWIDTH *CHARCODE FONT***)** [Function]

CHARCODE is an integer that represents a valid character (as returned by **CHCON1**). Returns the amount by which an image stream's X-position will be incremented when the character is printed.

(CHARWIDTHY *CHARCODE FONT***)** [Function]

Like **CHARWIDTH**, but returns the Y component of the character's width, the amount by which an image stream's Y-position will be incremented when the character is printed. This will be zero for most characters in normal portrait fonts, but may be non-zero for landscape fonts or for vector-drawing fonts.

(STRINGWIDTH *STR FONT FLG RDTBL***)** [Function]

Returns the amount by which a stream's X-position will be incremented if the printname for the Interlisp-D object *STR* is printed in font *FONT*. If *FONT* is an image stream, its font is used. If *FLG* is non-**NIL**, the **PRIN2**-pname of *STR* with respect to the readtable *RDTBL* is used.

(STRINGREGION *STR STREAM PRIN2FLG RDTBL***)** [Function]

Returns the region occupied by *STR* if it were printed at the current location in the image stream *STREAM*. This is useful, for example, for determining where text is in a window to allow the user to select it. The arguments *PRIN2FLG* and *RDTBL* are passed to **STRINGWIDTH**.

Note: **STRINGREGION** does not take into account any carriage returns in the string, or carriage returns that may be automatically printed if *STR* is printed to *STREAM*. Therefore, the value returned is meaningless for multi-line strings.

The following functions allow the user to access and change the bitmaps for individual characters in a display font. Note: Character code 256 can be used to access the "dummy" character, used for characters in the font with no bitmap defined.

(GETCHARBITMAP *CHARCODE FONT***)** [Function]

Returns a bitmap containing a copy of the image of the character *CHARCODE* in the font *FONT*.

(PUTCHARBITMAP *CHARCODE FONT NEWCHARBITMAP NEWCHARDESCENT***)** [Function]

Changes the bitmap image of the character *CHARCODE* in the font *FONT* to the bitmap *NEWCHARBITMAP*. If *NEWCHARDESCENT* is non-**NIL**, the descent of the character is changed to the value of *NEWCHARDESCENT*.

(**EDITCHAR** *CHARCODE FONT*) [Function]

Calls the bitmap editor (**EDITBM**, page 27.4) on the bitmap image of the character *CHARCODE* in the font *FONT*. *CHARCODE* can be a character code (as returned by **CHCON1**) or an atom or string, in which case the first character of *CHARCODE* is used.

27.13 Font Files and Font Directories

If **FONTCREATE** is called to create a font that has not been loaded into Interlisp, **FONTCREATE** has to read the font information from a font file that contains the information for that font. For printer devices, the font files have to contain width information for each character in the font. For display fonts, the font files have to contain, in addition, bitmap images for each character in the fonts. The font file names, formats, and searching algorithms are different for each device. There are a set of variables for each device, that determine the directories that are searched for font files. All of these variables must be set before Interlisp can auto-load font files. These variables should be initialized in the site-specific INIT file.

DISPLAYFONTDIRECTORIES [Variable]

Value is a list of directories searched to find font bitmap files for display fonts.

DISPLAYFONTEXTENSIONS [Variable]

Value is a list of file extensions used when searching **DISPLAYFONTDIRECTORIES** for display fonts. Initially set to **(DISPLAYFONT)**, but when using older font files it may be necessary to add **STRIKE** and **AC** to this list.

INTERPRESSFONTDIRECTORIES [Variable]

Value is a list of directories searched to find font widths files for Interpress fonts.

PRESSFONTWIDTHSFILES [Variable]

Value is a list of files (not directories) searched to find font widths files for Press fonts. Press font widths are packed into large files (usually named **FONTS.WIDTHS**).

27.15 Font Profiles

PRETTYPRINT contains a facility for printing different elements (user functions, system functions, clisp words, comments, etc.) in different fonts to emphasize (or deemphasize) their importance, and in general to provide for a more pleasing appearance. Of course, in order to be useful, this facility requires that the user is printing on a device (such as a bitmapped display or a laser printer) which supports multiple fonts.

PRETTYPRINT signals font changes by inserting into the file a user-defined escape sequence (the value of the variable **FONTESCAPECHAR**) followed by the character *code* which specifies, by number, which font to use, i.e. ↑**A** for font number 1, etc. Thus, if **FONTESCAPECHAR** were the character ↑**F**, ↑**F**↑**C** would be output to change to font 3, ↑**F**↑**A** to change to font 1, etc. If **FONTESCAPECHAR** consists of characters which are separator charactors in **FILERDTBL**, then a file with font changes in it can also be loaded back in.

Currently, **PRETTYPRINT** uses the following font classes. The user can specify separate fonts for each of these classes, or use the same font for several different classes.

LAMBDAFONT The font for printing the name of the function being prettyprinted, before the actual definition (usually a large font).

CLISPFONT If **CLISPFLG** is on, the font for printing any clisp words, i.e. atoms with property **CLISPWORD**.

COMMENTFONT The font used for comments.

USERFONT The font for the name of any function in the file, or any member of the list **FONTFNS**.

SYSTEMFONT The font for any other (defined) function.

CHANGEFONT The font for an expression marked by the editor as having been changed.

PRETTYCOMFONT The font for the operand of a file package command.

DEFAULTFONT The font for everything else.

Note that not all combinations of fonts will be aesthetically pleasing (or even readable!) and the user may have to experiment to find a compatible set.

Although in some implementations **LAMBDAFONT** et al. may be defined as variables, one should not set them directly, but should indicate what font is to be used for each class by calling the function **FONTPROFILE**:

(FONTPROFILE *PROFILE***)** [Function]

Sets up the font classes as determined by *PROFILE*, a list of elements which defines the correspondence between font

classes and specific fonts. Each element of *PROFILE* is a list of the form:

(*FONTCLASS FONT# DISPLAYFONT PRESSFONT INTERPRESSFONT*)

FONTCLASS is the font class name and *FONT#* is the font number for that class. For each font class name, the escape sequence will consist of **FONTESCAPECHAR** followed by the character *code* for the font number, e.g. ↑ **A** for font number 1, etc.

If *FONT#* is **NIL** for any font class, the font class named **DEFAULTFONT** (which must always be specified) is used. Alternatively, if *FONT#* is the name of a previously defined font class, this font class will be equivalenced to the previously defined one.

DISPLAYFONT, *PRESSFONT*, and *INTERPRESSFONT* are font specifications (of the form accepted by **FONTCREATE**) for the fonts to use when printing to the display and to Press and Interpress printers respectively.

FONTPROFILE [Variable]

This is the *variable* used to store the current font profile, in the form accepted by the *function* **FONTPROFILE**. Note that simply editing this value will not change the fonts used for the various font classes; it is necessary to execute **(FONTPROFILE FONTPROFILE)** to install the value of this variable.

The process of printing with multiple fonts is affected by a large number of variables: **FONTPROFILE**, **FILELINELENGTH**, **PRETTYLCOM**, etc. To facilitate switching back and forth between various sets of values for the font variables, Interlisp supports the idea of named "font configurations" encapsulating the values of all relevant variables.

To create a new font configuration, set all "relevant" variables to the values you want, and then call **FONTNAME** to save them (on the variable **FONTDEFS**) under a given name. To install a particular font configuration, call **FONTSET** giving it your name. To change the values in a saved font configuration, edit the value of the variable **FONTDEFS**.

Note: The list of variables saved by **FONTNAME** is stored in the variable **FONTDEFSVARS**. This can be changed by the user.

(FONTNAME *NAME***)** [Function]

Collects the names and values of the variables on **FONTDEFSVARS**, and saves them on **FONTDEFS**.

(FONTSET *NAME*) [Function]

Installs font configuration for *NAME*. Also evaluates **(FONTPROFILE FONTPROFILE)** to install the font classes as specified in the new value of the variable **FONTPROFILE**. Generates an error if *NAME* not previously defined.

FONTDEFSVARS [Variable]

The list of variables to be packaged by a **FONTNAME**. Initially **FONTCHANGEFLG**, **FILELINELENGTH**, **COMMENTLINELENGTH**, **FIRSTCOL**, **PRETTYLCOM**, **LISTFILESTR**, and **FONTPROFILE**.

FONTDEFS [Variable]

An association list of font configurations. **FONTDEFS** is a list of elements of form **(***NAME* . *PARAMETER-PAIRS***)**. To save a configuration on a file after performing a **FONTNAME** to define it, the user could either save the entire value of **FONTDEFS**, or use the **ALISTS** file package command (page 17.37) to dump out just the one configuration.

FONTESCAPECHAR [Variable]

The character or string used to signal the start of a font escape sequence.

FONTCHANGEFLG [Variable]

If **T**, enables fonts when prettyprinting. If **NIL**, disables fonts.

LISTFILESTR [Variable]

In Interlisp-10, passed to the operating system by **LISTFILES** (page 17.14). Can be used to specify subcommands to the **LIST** command, e.g. to establish correspondance between font number and font name.

COMMENTLINELENGTH [Variable]

Since comments are usually printed in a smaller font, **COMMENTLINELENGTH** is provided to offset the fact that Interlisp does not know about font widths. When **FONTCHANGEFLG** = **T**, **CAR** of **COMMENTLINELENGTH** is the linelength used to print short comments, i.e. those printed in the right margin, and **CDR** is the linelength used when printing full width comments.

(CHANGEFONT *FONT STREAM*) [Function]

Executes the operations on *STREAM* to change to the font *FONT*. For use in **PRETTYPRINTMACROS**.

27.16 Image Objects

An Image Object is an object that includes information about an image, such as how to display it, how to print it, and how to manipulate it when it is included in a collection of images (such as a document). More generally, it enables you to include one kind of image, with its own semantics, layout rules, and editing paradigms, inside another kind of image. Image Objects provide a general-purpose interface between image users who want to manipulate arbitrary images, and image producers, who create images for use, say, in documents.

Images are encapsulated inside a uniform barrier—the **IMAGEOBJ** data type. From the outside, you communicate to the image by calling a standard set of functions. For example, calling one function tells you how big the image is; calling another causes the image object to be displayed where you tell it, and so on. Anyone who wants to create images for general use can implement his own brand of **IMAGEOBJ**. **IMAGEOBJ**s have been implemented (in library packages) for bitmaps, menus, annotations, graphs, and sketches.

Image Objects were originally implemented to support inserting images into TEdit text files, but the facility is available for use by any tools that manipulate images. The Image Object interface allows objects to exist in TEdit documents and be edited with their own editor. It also provides a facility in which objects can be shift-selected (or "copy-selected") between TEdit and non-TEdit windows. For example, the Image Objects interface allows you to copy-select graphs from a Grapher window into a TEdit window. The source window (where the object comes from) does not have to know what sort of window the destination window (where the object is inserted) is, and the destination does not have to know where the insertion comes from.

A new data type, **IMAGEOBJ**, contains the data and the procedures necessary to manipulate an object that is to be manipulated in this way. **IMAGEOBJ**s are created with the function **IMAGEOBJCREATE** (below).

Another new data type, **IMAGEFNS**, is a vector of the procedures necessary to define the behavior of a type of **IMAGEOBJ**. Grouping the operations in a separate data type allows multiple instances of the same type of image object to share procedure vectors. The data and procedure fields of an **IMAGEOBJ** have a uniform interface through the function **IMAGEOBJPROP**. **IMAGEFNS** are created with the function **IMAGEFNSCREATE**:

(IMAGEFNSCREATE *DISPLAYFN IMAGEBOXFN PUTFN GETFN COPYFN BUTTONEVENTINFN COPYBUTTONEVENTINFN WHENMOVEDFN WHENINSERTEDFN WHENDELETEDFN WHENCOPIEDFN WHENOPERATEDONFN PREPRINTFN* —**)** [Function]

Returns an **IMAGEFNS** object that contains the functions necessary to define the behavior of an **IMAGEOBJ**.

The arguments *DISPLAYFN* through *PREPRINTFN* should all be function names to be stored as the "methods" of the **IMAGEFNS**. The purpose of each **IMAGEFNS** method is described below.

Note: Image objects must be "registered" before they can be read by TEdit or **HREAD** (see page 27.39). **IMAGEFNSCREATE** implicitly registers its *GETFN* argument.

(IMAGEOBJCREATE *OBJECTDATUM IMAGEFNS***)** [Function]

Returns an **IMAGEOBJ** that contains the object datum *OBJECTDATUM* and the operations vector *IMAGEFNS*. *OBJECTDATUM* can be arbitrary data.

(IMAGEOBJPROP *IMAGEOBJECT PROPERTY NEWVALUE***)** [NoSpread Function]

Accesses and sets the properties of an **IMAGEOBJ**. Returns the current value of the *PROPERTY* property of the image object *IMAGEOBJECT*. If *NEWVALUE* is given, the property is set to it.

IMAGEOBJPROP can be used on the system properties **OBJECTDATUM**, **DISPLAYFN**, **IMAGEBOXFN**, **PUTFN**, **GETFN**, **COPYFN**, **BUTTONEVENTINFN**, **COPYBUTTONEVENTINFN**, **WHENOPERATEDONFN**, and **PREPRINTFN**. Additionally, it can be used to save arbitrary properties on an **IMAGEOBJ**.

(IMAGEFNSP *X***)** [Function]

Returns *X* if *X* is an **IMAGEFNS** object, **NIL** otherwise.

(IMAGEOBJP *X***)** [Function]

Returns *X* if *X* is an **IMAGEOBJ** object, **NIL** otherwise.

27.16.1 IMAGEFNS Methods

Note: Many of the **IMAGEFNS** methods below are passed "host stream" arguments. The TEdit text editor passes the "text stream" (an object contain all of the information in the document being edited) as the "host stream" argument. Other editing programs that want to use image objects may want to pass the data structure being edited to the **IMAGEFNS** methods as the "host stream" argument.

(DISPLAYFN *IMAGEOBJ IMAGESTREAM IMAGESTREAMTYPE HOSTSTREAM***)** [IMAGEFNS Method]

The **DISPLAYFN** method is called to display the object *IMAGEOBJ* at the current position on *IMAGESTREAM*. The type of *IMAGESTREAM* indicates whether the device is the display or some other image stream.

Note: When the **DISPLAYFN** method is called, the offset and clipping regions for the stream are set so the object's image is at **(0,0)**, and only that image area can be modified.

(IMAGEBOXFN *IMAGEOBJ IMAGESTREAM CURRENTX RIGHTMARGIN***)** [IMAGEFNS Method]

The **IMAGEBOXFN** method should return the size of the object as an **IMAGEBOX**, which is a data structure that describes the image laid down when an **IMAGEOBJ** is displayed in terms of width, height, and descender height. An **IMAGEBOX** has four fields: **XSIZE**, **YSIZE**, **YDESC**, and **XKERN**. **XSIZE** and **YSIZE** are the width and height of the object image. **YDESC** and **XKERN** give the position of the baseline and the left edge of the image relative to where you want to position it. For characters, the **YDESC** is the descent (height of the descender) and the **XKERN** is the amount of left kerning (note: TEdit doesn't support left kerning).

The **IMAGEBOXFN** looks at the type of the stream to determine the output device if the object's size changes from device to device. (For example, a bit-map object may specify a scale factor that is ignored when the bit map is displayed on the screen.) *CURRENTX* and *RIGHTMARGIN* allow an object to take account of its environment when deciding how big it is. If these fields are not available, they are **NIL**.

Note: TEdit calls the **IMAGEBOXFN** only during line formatting, then caches the **IMAGEBOX** as the **BOUNDBOX** property of the **IMAGEOBJ**. This avoids the need to call the **IMAGEBOXFN** when incomplete position and margin information is available.

(PUTFN *IMAGEOBJ FILESTREAM***)** [IMAGEFNS Method]

The **PUTFN** method is called to save the object on a file. It prints a description on *FILESTREAM* that, when read by the corresponding **GETFN** method (see below), regenerates the image object. (TEdit and **HPRINT** take care of writing out the name of the **GETFN**.)

(GETFN *FILESTREAM***)** [IMAGEFNS Method]

The **GETFN** method is called when the object is encountered on the file during input. It reads the description that was written by the **PUTFN** method and returns an **IMAGEOBJ**.

(COPYFN *IMAGEOBJ SOURCEHOSTSTREAM TARGETHOSTSTREAM***)** [IMAGEFNS Method]

The **COPYFN** method is called during a copy-select operation. It should return a copy of *IMAGEOBJ*. If it returns the litatom **DON'T**, copying is suppressed.

(BUTTONEVENTINFN *IMAGEOBJ WINDOWSTREAM SELECTION RELX RELY WINDOW HOSTSTREAM BUTTON***)** [IMAGEFNS Method]

The **BUTTONEVENTINFN** method is called when you press a mouse button inside the object. The **BUTTONEVENTINFN** decides whether or not to handle the button, to track the cursor in parallel with mouse movement, and to invoke selections or edits supported by the object (but see the **COPYBUTTONEVENTINFN** method below). If the **BUTTONEVENTINFN** returns **NIL**, TEdit treats the button press as a selection at its level. Note that when this function is first called, a button is down. The **BUTTONEVENTINFN** should also support the button-down protocol to descend inside of any composite objects with in it. In most cases, the **BUTTONEVENTINFN** relinquishes control (i.e., returns) when the cursor leaves its object's region.

Note: When the **BUTTONEVENTINFN** is called, the window's clipping region and offsets have been changed so that the lower-left corner of the object's image is at **(0,0)**, and only the object's image can be changed. The selection is available for changing to fit your needs; the mouse button went down at **(***RELX,RELY***)** within the object's image. You can affect how TEdit treats the selection by returning one of several values. If you return **NIL**, TEdit forgets that you selected an object; if you return the atom **DON'T**, TEdit doesn't permit the selection; if you return the atom **CHANGED**, TEdit updates the screen. Use **CHANGED** to signal TEdit that the object has changed size or will have side effects on other parts of the screen image.

(COPYBUTTONEVENTINFN *IMAGEOBJ WINDOWSTREAM***)** [IMAGEFNS Method]

The **COPYBUTTONEVENTINFN** method is called when you button inside an object while holding down a copy key. Many of the comments about **BUTTONEVENTINFN** apply here too. Also, see the discussion below about copying image objects between windows (page 27.41).

(WHENMOVEDFN *IMAGEOBJ TARGETWINDOWSTREAM SOURCEHOSTSTREAM TARGETHOSTSTREAM***)** [IMAGEFNS Method]

The **WHENMOVEDFN** method provides hooks by which the object is notified when TEdit performs an operation (MOVEing) on the whole object. It allows objects to have side effects.

(WHENINSERTEDFN *IMAGEOBJ TARGETWINDOWSTREAM SOURCEHOSTSTREAM TARGETHOSTSTREAM***)** [IMAGEFNS Method]

The **WHENINSERTEDFN** method provides hooks by which the object is notified when TEdit performs an operation (INSERTing) on the whole object. It allows objects to have side effects.

(WHENDELETEDFN *IMAGEOBJ TARGETWINDOWSTREAM***)** [IMAGEFNS Method]

The **WHENDELETEDFN** method provides hooks by which the object is notified when TEdit performs an operation (DELETEing) on the whole object. It allows objects to have side effects.

(WHENCOPIEDFN *IMAGEOBJ TARGETWINDOWSTREAM SOURCEHOSTSTREAM TARGETHOSTSTREAM***)** [IMAGEFNS Method]

The **WHENCOPIEDFN** method provides hooks by which the object is notified when TEdit performs an operation (COPYing) on the whole object. The **WHENCOPIEDFN** method is called in addition to (and after) the **COPYFN** method above. It allows objects to have side effects.

(WHENOPERATEDONFN *IMAGEOBJ WINDOWSTREAM HOWOPERATEDON SELECTION HOSTSTREAM***)** [IMAGEFNS Method]

The **WHENOPERATEDONFN** method provides a hook for edit operations. *HOWOPERATEDON* should be one of **SELECTED**, **DESELECTED**, **HIGHLIGHTED**, and **UNHILIGHTED**. The **WHENOPERATEDONFN** differs from the **BUTTONEVENTINFN** because it is called when you extend a selection through the object. That is, the object is treated in toto as a TEdit character. **HIGHLIGHTED** refers to the selection being highlighted on the screen, and **UNHIGHLIGHTED** means that the highlighting is being turned off.

(PREPRINTFN *IMAGEOBJ***)** [IMAGEFNS Method]

The **PREPRINTFN** method is called to convert the object into something that can be printed for inclusion in documents. It returns an object that the receiving window can print (using either **PRIN1** or **PRIN2**,its choice) to obtain a character representation of the object. If the **PREPRINTFN** method is **NIL**, the **OBJECTDATUM** field of *IMAGEOBJ* itself is used. TEdit uses this function when you indicate that you want to print the characters from an object rather than the object itself (presumably using **PRIN1** case).

27.16.2 Registering Image Objects

Each legitimate **GETFN** needs to be known to the system, to prevent various Trojan-horse problems and to allow the

automatic loading of the supporting code for infrequently used **IMAGEOBJ**s. To this end, there is a global list, **IMAGEOBJGETFNS**, that contains an entry for each **GETFN**. The existence of the entry marks the **GETFN** as legitimate; the entry itself is a property list, which can hold information about the **GETFN**.

No action needs to be taken for **GETFN**s that are currently in use: the function **IMAGEFNSCREATE** automatically adds its *GETFN* argument to the list. However, packages that support obsolete versions of objects may need to explicitly add the obsolete **GETFN**s. For example, TEdit supports bit-map **IMAGEOBJ**s. Recently, a change was made in the format in which objects are stored; to retain compatibility with the old object format, there are now two **GETFN**s. The current **GETFN** is automatically on the list, courtesy of **IMAGEFNSCREATE**. However, the code file that supports the old bit-map objects contains the clause: **(ADDVARS (IMAGEOBJGETFNS (***OLDGETFNNAME***)))**, which adds the old **GETFN** to **IMAGEOBJGETFNS**.

For a given **GETFN**, the entry on **IMAGEOBJGETFNS** may be a property list of information. Currently the only recognized property is **FILE**.

FILE is the name of the file that can be loaded if the **GETFN** isn't defined. This file should define the **GETFN**, along with all the other functions needed to support that kind of **IMAGEOBJ**.

For example, the bit-map **IMAGEOBJ** implemented by TEdit use the **GETFN BMOBJ.GETFN2**. Its entry on **IMAGEOBJGETFNS** is **(BMOBJ.GETFN2 FILE IMAGEOBJ)**, indicating that the support code for bit-map image objects resides on the file **IMAGEOBJ**, and that the **GETFN** for them is **BMOBJ.GETFN2**.

This makes it possible to have entries for **GETFN**s whose supporting code isn't loaded—you might, for instance, have your init file add entries to **IMAGEOBJGETFNS** for the kinds of image objects you commonly use. The system's default reading method will automatically load the code when necessary.

27.16.3 Reading and Writing Image Objects on Files

Image Objects can be written out to files using **HPRINT** and read back using **HREAD**. The following functions can also be used:

(WRITEIMAGEOBJ *IMAGEOBJ STREAM***)** [Function]

Prints (using **PRIN2**) a call to **READIMAGEOBJ**, then calls the **PUTFN** for *IMAGEOBJ* to write it onto *STREAM*. During input, then, the call to **READIMAGEOBJ** is read and evaluated; it in turn reads back the object's description, using the appropriate **GETFN**.

(READIMAGEOBJ *STREAM GETFN NOERROR***)** [Function]

Reads an **IMAGEOBJ** from *STREAM*, starting at the current file position. Uses the function *GETFN* after validating it (and loading support code, if necessary).

If the *GETFN* can't be validated or isn't defined, **READIMAGEOBJ** returns an "encapsulated image object", an **IMAGEOBJ** that safely encapsulates all of the information in the image object. An encapsulated image object displays as a rectangle that says, "**Unknown IMAGEOBJ Type**" and lists the *GETFN*'s name. Selecting an encapsulated image object with the mouse causes another attempt to read the object from the file; this is so you can load any necessary support code and then get to the object.

Warning: You cannot save an encapsulated image object on a file because there isn't enough information to allow copying the description to the new file from the old one.

If *NOERROR* is non-**NIL**, **READIMAGEOBJ** returns **NIL** if it can't successfully read the object.

27.16.4 Copying Image Objects Between Windows

Copying between windows is implemented as follows: If a button event occurs in a window when a copy key is down, the window's **COPYBUTTONEVENTFN** window property is called. If this window supports copy-selection, it should track the mouse, indicating the item to be copied. When the button is released, the **COPYBUTTONEVENTFN** should create an image object out of the selected information, and call **COPYINSERT** to insert it in the current TTY window. **COPYINSERT** calls the **COPYINSERTFN** window property of the TTY window to insert this image object. Therefore, both the source and destination windows can determine how they handle copying image objects.

If the **COPYBUTTONEVENTFN** of a window is **NIL**, the **BUTTONEVENTFN** is called instead when a button event occurs in the window when a copy key is down, and copying from that window is not supported. If the **COPYINSERTFN** of the TTY window is **NIL**, **COPYINSERT** will turn the image object into a string (by calling the **PREPRINTFN** method of the image object, see page 27.39) and insert it by calling **BKSYSBUF** (page 30.11).

COPYBUTTONEVENTFN [Window Property]

The **COPYBUTTONEVENTFN** of a window is called (if it exists) when a button event occurs in the window and a copy key is down. If no **COPYBUTTONEVENTFN** exists, the **BUTTONEVENTFN** is called.

COPYINSERTFN [Window Property]

The **COPYINSERTFN** of the "destination" window is called by **COPYINSERT** to insert something into the destination window. It is called with two arguments: the object to be inserted and the destination window. The object to be inserted can be a character string, an **IMAGEOBJ**, or a list of **IMAGEOBJ**s and character strings. As a convention, the **COPYINSERTFN** should call **BKSYSBUF** (page 30.11) if the object to be inserted insert is a character string.

(COPYINSERT *IMAGEOBJ***)** [Function]

COPYINSERT inserts *IMAGEOBJ* into the window that currently has the TTY. If the current TTY window has a **COPYINSERTFN**, it is called, passing it *IMAGEOBJ* and the window as arguments.

If no **COPYINSERTFN** exists and if *IMAGEOBJ* is an image object, **BKSYSBUF** is called on the result of calling its **PREPRINTFN** on it. If *IMAGEOBJ* is not an image object, it is simply passed to **BKSYSBUF** (page 30.11). In this case, **BKSYSBUF** will call **PRIN2** with a read table taken from the process associated with the TTY window. A window that wishes to use **PRIN1** or a different read table must provide its own **COPYINSERTFN** to do this.

27.17 Implementation of Image Streams

Interlisp does all image creation through a set of functions and data structures for device-independent graphics, known popularly as DIG. DIG is implemented through the use of a special type of stream, known as an image stream.

An image stream, by convention, is any stream that has its **IMAGEOPS** field (described in detail below) set to a vector of meaningful graphical operations. Using image streams, you can write programs that draw and print on an output stream without regard to the underlying device, be it a window, a disk, or a printer.

To define a new image stream type, it is necessary to put information on the variable **IMAGESTREAMTYPES**:

IMAGESTREAMTYPES [Variable]

This variable describes how to create a stream for a given image stream type. The value of **IMAGESTREAMTYPES** is an association list, indexed by the image stream type (e.g., **DISPLAY**, **INTERPRESS**, etc.). The format of a single association list item is:

(*IMAGETYPE*
 (OPENSTREAM *OPENSTREAMFN***)**

(FONTCREATE *FONTCREATEFN***)**
(FONTSAVAILABLE *FONTSAVAILABLEFN***))**

OPENSTREAMFN, *FONTCREATEFN*, and *FONTSAVAILABLEFN* are "image stream methods," device-dependent functions used to implement generic image stream operations. For Interpress image streams, the association list entry is:

```
(INTERPRESS
  (OPENSTREAM OPENIPSTREAM)
  (FONTCREATE \CREATEINTERPRESSFONT)
  (FONTSAVAILABLE \SEARCHINTERPRESSFONTS))
```

(OPENSTREAMFN *FILE OPTIONS***)** [Image Stream Method]

FILE is the file name as it was passed to **OPENIMAGESTREAM**, and *OPTIONS* is the *OPTIONS* property list passed to **OPENIMAGESTREAM**. The result must be a stream of the appropriate image type.

(FONTCREATEFN *FAMILY SIZE FACE ROTATION DEVICE***)** [Image Stream Method]

FAMILY is the family name for the font, e.g., **MODERN**. *SIZE* is the body size of the font, in printer's points. *FACE* is a three-element list describing the weight, slope, and expansion of the face desired, e.g., **(MEDIUM ITALIC EXPANDED)**. *ROTATION* is how much the font is to be rotated from the normal orientation, in minutes of arc. For example, to print a landscape page, fonts have the rotation 5400 (90 degrees). The function's result must be a **FONTDESCRIPTOR** with the fields filled in appropriately.

(FONTSAVAILABLEFN *FAMILY SIZE FACE ROTATION DEVICE***)** [Image Stream Method]

This function returns a list of all fonts agreeing with the *FAMILY*, *SIZE*, *FACE*, and *ROTATION* arguments; any of them may be wild-carded (i.e., equal to *, which means any value is acceptable). Each element of the list should be a quintuple of the form **(***FAMILY SIZE FACE ROTATION DEVICE***)**.

Where the function looks is an implementation decision: the *FONTSAVAILABLEFN* for the display device looks at **DISPLAYFONTDIRECTORIES**, the Interpress code looks on **INTERPRESSFONTDIRECTORIES**, and implementors of new devices should feel free to introduce new search path variables.

As indicated above, image streams use a field that no other stream uses: **IMAGEOPS**. **IMAGEOPS** is an instance of the **IMAGEOPS** data type and contains a vector of the stream's graphical methods. The methods contained in the **IMAGEOPS** object can make arbitrary use of the stream's **IMAGEDATA** field,

which is provided for their use, and may contain any data needed.

The **IMAGEOPS** data type has the following fields:

IMAGETYPE [IMAGEOPS Field]

Value is the name of an image type. Monochrome display streams have an **IMAGETYPE** of **DISPLAY**; color display streams are identified as **(COLOR DISPLAY)**. The **IMAGETYPE** field is informational and can be set to anything you choose.

IMFONTCREATE [IMAGEOPS Field]

Value is the device name to pass to **FONTCREATE** when fonts are created for the stream.

The remaining fields are all image stream methods, whose value should be a device-dependent function that implements the generic operation. Most methods are called by a similarly-named function, e.g. the function **DRAWLINE** calls the **IMDRAWLINE** method. All coordinates that refer to points in a display device's space are measured in the device's units. (The **IMSCALE** method provides access to a device's scale.) For arguments that have defaults (such as the *BRUSH* argument of **DRAWCURVE**), the default is substituted for the **NIL** argument before it is passed to the image stream method. Therefore, image stream methods do not have to handle defaults.

(IMCLOSEFN *STREAM***)** [Image Stream Method]

Called before a stream is closed with **CLOSEF**. This method should flush buffers, write header or trailer information, etc.

(IMDRAWLINE *STREAM* X_1 Y_1 X_2 Y_2 *WIDTH OPERATION COLOR DASHING***)** [Image Stream Method]

Draws a line of width *WIDTH* from (X_1, Y_1) to (X_2, Y_2). See **DRAWLINE**, page 27.17.

(IMDRAWCURVE *STREAM KNOTS CLOSED BRUSH DASHING***)** [Image Stream Method]

Draws a curve through *KNOTS*. See **DRAWCURVE**, page 27.19.

(IMDRAWCIRCLE *STREAM CENTERX CENTERY RADIUS BRUSH DASHING***)** [Image Stream Method]

Draws a circle of radius *RADIUS* around *(CENTERX, CENTERY)*. See **DRAWCIRCLE**, page 27.19.

(IMDRAWELLIPSE *STREAM CENTERX CENTERY SEMIMINORRADIUS SEMIMAJORRADIUS ORIENTATION BRUSH DASHING***)** [Image Stream Method]

Draws an ellipse around (*CENTERX*, *CENTERY*). See **DRAWELLIPSE**, page 27.19.

(IMFILLPOLYGON *STREAM POINTS TEXTURE***)** [Image Stream Method]

Fills in the polygon outlined by *POINTS* on the image stream *STREAM*, using the texture *TEXTURE*. See **FILLPOLYGON**, page 27.20.

(IMFILLCIRCLE *STREAM CENTERX CENTERY RADIUS TEXTURE***)** [Image Stream Method]

Draws a circle filled with texture *TEXTURE* around (*CENTERX*, *CENTERY*). See **FILLCIRCLE**, page 27.21.

(IMBLTSHADE *TEXTURE STREAM DESTINATIONLEFT DESTINATIONBOTTOM WIDTH HEIGHT OPERATION CLIPPINGREGION***)** [Image Stream Method]

The texture-source case of **BITBLT** (page 27.14). *DESTINATIONLEFT*, *DESTINATIONBOTTOM*, *WIDTH*, *HEIGHT*, and *CLIPPINGREGION* are measured in *STREAM*'s units. This method is invoked by the functions **BITBLT** and **BLTSHADE** (page 27.16).

(IMBITBLT *SOURCEBITMAP SOURCELEFT SOURCEBOTTOM STREAM DESTINATIONLEFT DESTINATIONBOTTOM WIDTH HEIGHT SOURCETYPE OPERATION TEXTURE CLIPPINGREGION CLIPPEDSOURCELEFT CLIPPEDSOURCEBOTTOM SCALE***)** [Image Stream Method]

Contains the bit-map-source cases of **BITBLT** (page 27.14). *SOURCELEFT*, *SOURCEBOTTOM*, *CLIPPEDSOURCELEFT*, *CLIPPEDSOURCEBOTTOM*, *WIDTH*, and *HEIGHT* are measured in pixels; *DESTINATIONLEFT*, *DESTINATIONBOTTOM*, and *CLIPPINGREGION* are in the units of the destination stream.

(IMSCALEDBITBLT *SOURCEBITMAP SOURCELEFT SOURCEBOTTOM STREAM DESTINATIONLEFT DESTINATIONBOTTOM WIDTH HEIGHT SOURCETYPE OPERATION TEXTURE CLIPPINGREGION CLIPPEDSOURCELEFT CLIPPEDSOURCEBOTTOM SCALE***)** [Image Stream Method]

A scaled version of **IMBITBLT**. Each pixel in *SOURCEBITMAP* is replicated *SCALE* times in the X and Y directions; currently, *SCALE* must be an integer.

(IMMOVETO *STREAM X Y***)** [Image Stream Method]

Moves to (*X*,*Y*). This method is invoked by the function **MOVETO** (page 27.13). If **IMMOVETO** is not supplied, a default method composed of calls to the **IMXPOSITION** and **IMYPOSITION** methods is used.

(IMSTRINGWIDTH *STREAM STR RDTBL***)** [Image Stream Method]

Returns the width of string *STR* in *STREAM*'s units, using *STREAM*'s current font. This is envoked when **STRINGWIDTH** (page 27.30) is passed a stream as its *FONT* argument. If **IMSTRINGWIDTH** is not supplied, it defaults to calling **STRINGWIDTH** on the default font of *STREAM*.

(IMCHARWIDTH *STREAM CHARCODE***)** [Image Stream Method]

Returns the width of character *CHARCODE* in *STREAM*'s units, using *STREAM*'s current font. This is invoked when **CHARWIDTH** (page 27.30) is passed a stream as its *FONT* argument. If **IMCHARWIDTH** is not supplied, it defaults to calling **CHARWIDTH** on the default font of *STREAM*.

(IMCHARWIDTHY *STREAM CHARCODE***)** [Image Stream Method]

Returns the Y componant of the width of character *CHARCODE* in *STREAM*'s units, using *STREAM*'s current font. This is envoked when **CHARWIDTHY** (page 27.30) is passed a stream as its *FONT* argument. If **IMCHARWIDTHY** is not supplied, it defaults to calling **CHARWIDTHY** on the default font of *STREAM*.

(IMBITMAPSIZE *STREAM BITMAP DIMENSION***)** [Image Stream Method]

Returns the size that *BITMAP* will be when **BITBLT**ed to *STREAM*, in *STREAM*'s units. *DIMENSION* can be one of **WIDTH**, **HEIGHT**, or **NIL**, in which case the dotted pair (*WIDTH* . *HEIGHT*) will be returned.

This is envoked by **BITMAPIMAGESIZE** (page 27.16). If **IMBITMAPSIZE** is not supplied, it defaults to a method that multiplies the bitmap height and width by the scale of *STREAM*.

(IMNEWPAGE *STREAM***)** [Image Stream Method]

Causes a new page to be started. The X position is set to the left margin, and the Y position is set to the top margin plus the linefeed. If not supplied, defaults to **(\OUTCHAR** *STREAM* **(CHARCODE ↑ L))**. Envoked by **DSPNEWPAGE** (page 27.21).

(IMTERPRI *STREAM***)** [Image Stream Method]

Causes a new line to be started. The X position is set to the left margin, and the Y position is set to the current Y position plus the linefeed. If not supplied, defaults to **(\OUTCHAR** *STREAM* **(CHARCODE EOL))**. Envoked by **TERPRI** (page 25.9).

(IMRESET *STREAM***)** [Image Stream Method]

Resets the X and Y position of *STREAM*. The X coordinate is set to its left margin; the Y coordinate is set to the top of the

clipping region minus the font ascent. Envoked by **DSPRESET**, page 27.21.

The following methods all have corresponding **DSP***xx* functions (e.g., **IMYPOSITION** corresponds to **DSPYPOSITION**) that invoke them. They also have the property of returning their previous value; when called with **NIL** they return the old value without changing it.

(IMCLIPPINGREGION *STREAM REGION***)** [Image Stream Method]

Sets a new clipping region on *STREAM*.

(IMXPOSITION *STREAM XPOSITION***)** [Image Stream Method]

Sets the X-position on *STREAM*.

(IMYPOSITION *STREAM YPOSITION***)** [Image Stream Method]

Sets a new Y-position on *STREAM*.

(IMFONT *STREAM FONT***)** [Image Stream Method]

Sets *STREAM*'s font to be *FONT*.

(IMLEFTMARGIN *STREAM LEFTMARGIN***)** [Image Stream Method]

Sets *STREAM*'s left margin to be *LEFTMARGIN*. The left margin is defined as the X-position set after the new line.

(IMRIGHTMARGIN *STREAM RIGHTMARGIN***)** [Image Stream Method]

Sets *STREAM*'s right margin to be *RIGHTMARGIN*. The right margin is defined as the maximum X-position at which characters are printed; printing beyond it causes a new line.

(IMTOPMARGIN *STREAM YPOSITION***)** [Image Stream Method]

Sets *STREAM*'s top margin (the Y-position of the tops of characters that is set after a new page) to be *YPOSITION*.

(IMBOTTOMMARGIN *STREAM YPOSITION***)** [Image Stream Method]

Sets *STREAM*'s bottom margin (the Y-position beyond which any printing causes a new page) to be *YPOSITION*.

(IMLINEFEED *STREAM DELTA***)** [Image Stream Method]

Sets *STREAM*'s line feed distance (distance to move vertically after a new line) to be *DELTA*.

(IMSCALE *STREAM SCALE***)** [Image Stream Method]

Returns the number of device points per screen point (a screen point being ~1/72 inch). *SCALE* is ignored.

(IMSPACEFACTOR *STREAM FACTOR***)** [Image Stream Method]

Sets the amount by which to multiply the natural width of all following space characters on *STREAM*; this can be used for the justification of text. The default value is 1. For example, if the natural width of a space in *STREAM*'s current font is 12 units, and the space factor is set to two, spaces appear 24 units wide. The values returned by **STRINGWIDTH** and **CHARWIDTH** are also affected.

(IMOPERATION *STREAM OPERATION***)** [Image Stream Method]

Sets the default **BITBLT** *OPERATION* argument (see page 27.15).

(IMBACKCOLOR *STREAM COLOR***)** [Image Stream Method]

Sets the background color of *STREAM*.

(IMCOLOR *STREAM COLOR***)** [Image Stream Method]

Sets the default color of *STREAM*.

In addition to the **IMAGEOPS** methods described above, there are two other important methods, which are contained in the stream itself. These fields can be installed using a form like **(replace (STREAM OUTCHARFN) of STREAM with (FUNCTION MYOUTCHARFN))**. Note: You need to have loaded the Interlisp-D system declarations to manipulate the fields of **STREAM**s. The declarations can be loaded by loading the Lisp Library package **SYSEDIT**.

(STRMBOUTFN *STREAM CHARCODE***)** [Stream Method]

The function called by **BOUT**.

(OUTCHARFN *STREAM CHARCODE***)** [Stream Method]

The function that is called to output a single byte. This is like **STRMBOUTFN**, except for being one level higher: it is intended for text output. Hence, this function should convert **(CHARCODE EOL)** into the stream's actual end-of-line sequence and should adjust the stream's **CHARPOSITION** appropriately before invoking the stream's **STRMBOUTFN** (by calling **BOUT**) to actually put the character. Defaults to **\FILEOUTCHARFN**, which is probably incorrect for an image stream.