

## CHAPTER 19

### INTERLISP-D DISPLAY FACILITIES

This chapter describes the functions that support the display and the interaction with programs that use the display. First, a brief introductory view of using the Interlisp-D display and how some of the other Interlisp facilities have been extended to include display interfaces. The two screen images at left show some of the display features as used by exploratory programming tools of the Interlisp-D environment. The screen is divided into several rectangular areas or windows, each of which provides a view onto some data or process and which can be reshaped and repositioned at will by the user. When they overlap, the occluded portion of the lower window is automatically saved, so that it can be restored when the overlapping window is removed. Since the display is bitmapped, each window can contain an arbitrary mixture of text, lines, curves, and half-tone and solid area images.

The typescript window is in the upper left corner of the screen. It corresponds to the output channel T. In it, the user has defined a program F (factorial) and has then immediately run it, giving an input of 4 and getting a result of 24. Next, he queries the state of his lisp using the le package function FILES?, finding that one le has been changed (previously) and one function (F) has been defined but not associated with any le yet. The user sets the value of DRAWBETWEEN to 0 in command 74, and the system notes that this is a change and adds DRAWBETWEEN to the set of “changed objects” that might need to be saved.

Then, the user runs his program EDITTREE, giving it a parse tree for the sentence “My uncle’s story about the war will bore you to tears”. This opens up the big window on the right in which the sentence diagram is drawn. Using the mouse, the user starts to move the NP node on the left (which is inverted to show that it is being moved). While the move is taking place, the user interrupts the tree editor using Control-H, which suspends the computation and causes three “break” windows to appear on top of the lower edge of the typescript. These are part of the window break package. The smallest window shows the dynamic state of the computation, which has been broken inside a subprogram called FOLLOW/CURSOR. The “FOLLOW/CURSOR Frame” window to the right shows the value of the local variables bound by FOLLOW/CURSOR. One of them has been selected (and so appears inverted) and in response, its value has been shown in more detail in the window at the lower left of the screen. The user has marked one of the component values as suspicious by drawing on it using the window command PAINT. In addition, he has asked to examine the contents of the BITMAP component, which used the function EDITBM to open a bitmap edit window to the right. This shows an enlarged copy of the actual NP image that is being moved by the tree editor.

Inside the largest break window, the user has asked some questions about FOLLOW/CURSOR, and queried the value of DRAWBETWEEN (now 66). Using the BROWSER lispusers package, the Masterscope SHOW PATHS command brought up the horizontal tree diagram on the left, which shows which subprograms call each other, starting at FOLLOW/CURSOR. Each node in the call tree produced by the SHOW PATHS command is an active element which will respond to the user’s selecting it with the mouse. In the second image, the user has selected the SHOWNODE subprogram, which has caused its code to be retrieved from the le (<LISP>DEMO>LATTICER) on the remote le server (PHYLUM) where it was stored and displayed in the “Browser printout window” which has been opened at middle right. User programs and extended Lisp forms (like for and do) are highlighted by system generated font changes. By selecting nodes in the SHOW PATHS window, the user could also have edited or obtained a summary description of any of the

## POSITION

subprograms.

Instead, the user told Masterscope (in the break typescript window) to edit wherever anyone calls the DRAWBETWEEN program (a line drawing function). This request causes the system to consult its (dynamically maintained) database of information about user programs, wherein it finds that the subprogram SHOWLINK calls DRAWBETWEEN. It therefore loads the code for SHOWLINK into an edit window which appears under the "Browser print out window". The system then automatically finds and underlines the first (and only) call on DRAWBETWEEN. On the previous line, DRAWBETWEEN is used as a variable (the one the user set and interrogated earlier). The system, however, knows that this is not a subprogram call, so it has been skipped. If the user makes any change to SHOWLINK in the editor, not only will the change take effect immediately, but SHOWLINK will be marked as needing to be updated in its file and the information about it in the program database will be updated. This, in turn, will cause the SHOW PATHS window to be repainted, as its display may no longer be valid.

The Interlisp-D display facility has several layers. At the lowest level are routines which view the display as a collection of bits and provides primitives for moving blocks of bits around (BITBLT). The concepts important to this level are positions, regions and bitmaps. The next level is the display stream, an abstraction that implements clipping to rectangular areas of the screen, line and curve drawing, and printing to the screen in different fonts. The concepts important to this level are fonts and display streams. On the input side, there is a low level interface for reading the display input devices, the cursor location and the mouse buttons. The input and output come together at the next level, the window system which allows areas of the screen used by different programs to overlap by keeping track of information covered and providing control primitives for mouse interaction. This chapter is organized according to these levels.

### 19.1 POSITION

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.

### 19.2 REGION

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 returns 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

## INTERLISP-D DISPLAY FACILITIES

HEIGHT as respectively its LEFT, BOTTOM, WIDTH, and HEIGHT.

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 (10,-20).

(INTERSECTREGIONS REGION<sub>1</sub> REGION<sub>2</sub> ... REGION<sub>n</sub>) [NoSpread Function]  
Returns a region which is the intersection of a number of regions. Returns NIL if the intersection is empty. If there are no regions given, it returns a very large region.

(UNIONREGIONS REGION<sub>1</sub> REGION<sub>2</sub> ... REGION<sub>n</sub>) [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 LAR GEREGION SMALLREGION ) [Function]  
Returns T if SMALLREGION is a subregion (is equal to or entirely contained in) LAR GEREGION ; otherwise returns NIL.

(EXTENDREGION REGION INCLUDEREGION ) [Function]  
Changes (destructively modifies) the region REGION so that it includes the region INCLUDEREGION . It returns REGION .

(INSIDEP REGION X Y ) [Function]  
If x and y are numbers, it returns T if the point (x,y) is inside of REGION . If x is a POSITION, it returns T if x is inside of REGION . Otherwise, it returns NIL.

### 19.3 BITMAP

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 with the function VIDEOCOLOR; see page 19.7.) 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 with fields BITMAPWIDTH, BITMAPHEIGHT, BITMAPBITSPIXEL, BITMAPRASTERWIDTH, and BITMAPBASE. Only the width, height, and bits per pixel fields are of interest to the user, and can be accessed with the following functions:

(BITMAPWIDTH BITMAP ) [Function]  
Returns the width of BITMAP in pixels.

## BITBLT

(BITMAPHEIGHT BITMAP ) [Function]  
Returns the height of BITMAP in pixels.

(BITSPERPIXEL BITMAP ) [Function]  
Returns the number of bits per pixel of BITMAP .

The functions used to manipulate bitmaps are:

(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, the default is 1.

(BITMAPBIT BITMAP X Y NEWV ALUE ) [Function]  
If NEWV ALUE is between 0 and the maximum value for a pixel in BITMAP , the pixel (X,Y) is changed to NEWV ALUE and the old value is returned. If NEWV ALUE is NIL, BITMAP is not changed but the value of the pixel is returned. If NEWV ALUE 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.

(BITMAPCOPY BITMAP ) [Function]  
Returns a new bitmap which is a copy of BITMAP (same dimensions and contents).

(EXPANDBITMAP BITMAP WIDTHF ACTOR HEIGHTF ACTOR ) [Function]  
Returns a new bitmap that is WIDTHF ACTOR times as wide as BITMAP and HEIGHTF ACTOR times as high. Each pixel of BITMAP is copied into a WIDTHF ACTOR times HEIGHTF ACTOR block of pixels. If NIL, WIDTHF ACTOR defaults to 4, HEIGHTF ACTOR to 1.

There are two distinguished bitmaps that are read by the hardware to become visible as the screen and the cursor. The screen is a bitmap SCREENWIDTH (=1024) wide by SCREENHEIGHT (=808) high. The cursor is a bitmap CURSORWIDTH (=16) wide by CURSORHEIGHT (=16) high. They are accessed by:

(SCREENBITMAP) [Function]  
Returns the screen bitmap.

(CURSORBITMAP) [Function]  
Returns the cursor bitmap.

Note: The cursor bitmap can be changed with the function CURSOR (page 19.16).

### 19.4 BITBLT

BITBLT is the primitive function for moving bits from one bitmap to another. It is similar to the function RASTEROP that is used in other systems.

(BITBLT SOUR CEBITMAP SOUR CELEFT SOUR CEBOTTOM DESTINATIONBITMAP DESTINATIONLEFT  
DESTINATIONBOTTOM WIDTH HEIGHT SOUR CETYPE OPERATION TEXTURE CLIPPINGREGION ) [Function]

WIDTH and HEIGHT define a pair of rectangles, one in each of the SOUR CEBITMAP and DESTINATIONBITMAP

## INTERLISP-D DISPLAY FACILITIES

whose left, bottom corners are at, respectively, (SOURCELEFT, SOURCEBOTTOM) and (DESTINATIONLEFT, DESTINATIONBOTTOM). If these rectangles overlap the boundaries of either bitmap 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 DESTINATIONBITMAP; clipping to this region may further reduce the defining rectangles. These (possibly reduced) rectangles define the source and destination rectangles for BITBLT. SOURCEBITMAP and DESTINATIONBITMAP can also be display streams or windows, in which case their associated bitmaps are used.

The mode of transferring bits is defined by SOURCECETYPE and OPERATION. SOURCECETYPE and OPERATION specify boolean functions that are used to determine, respectively, the method of combining SOURCEBITMAP bits with the TEXTURE and the operation between these resultant bits and DESTINATIONBITMAP. TEXTURE is a gray pattern, as described on page 19.6. (Note: The alignment of the texture pattern with BITBLT is such that the origin of the destination bitmap is at an intersection of the “tiles.”)

SOURCECETYPE specifies how to combine the bits from SOURCEBITMAP with the bits from TEXTURE (a background pattern) to produce a “Source”. This is designed to allow characters and figures to be placed on a background.

SOURCECETYPE	Source
INPUT	SOURCEBITMAP
INVERT	(NOT SOURCEBITMAP)
TEXTURE	TEXTURE

For the INPUT and INVERT case, the TEXTURE argument to BITBLT is ignored. For the TEXTURE case, the SOURCEBITMAP, SOURCELEFT, and SOURCEBOTTOM arguments are ignored.

OPERATION specifies how this source is combined with the bits in DESTINATIONBITMAP and stored back into DESTINATIONBITMAP.

OPERATION	DESTINATIONBITMAP becomes
REPLACE	Source
PAINT	(OR DESTINATIONBITMAP Source)
INVERT	(XOR DESTINATIONBITMAP Source)
ERASE	(AND DESTINATIONBITMAP (NOT Source))

SOURCELEFT, SOURCEBOTTOM, DESTINATIONLEFT, and DESTINATIONBOTTOM default to 0. WIDTH and HEIGHT default to the width and height of the SOURCEBITMAP. TEXTURE defaults to white. SOURCECETYPE 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: BITBLT and BITMAPBIT accept windows and display streams as their bitmap arguments. In these cases, the remaining arguments are interpreted as values in the coordinate system of the window or display stream and the operation of the functions are translated and clipped accordingly. If a window or display stream is used as the destination to BITBLT, its clipping region limits the operation involved.

## TEXTURE

### 19.5 TEXTURE

A Texture denotes a pattern of gray which can be used by BITBLT to (conceptually) tessellate the plane to form an infinite sheet of gray. It is currently a 4 by 4 pattern. Textures are created interactively using the function EDITSHADE or 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, i.e. a legal texture argument to BITBLT.

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. The original background shade of the window system is kept in WINDOWBACKGROUNDSHADE . The background shade can be changed by the following function:

(CHANGEBACKGROUND SHADE ) [Function]  
Changes the background shade of the window system. SHADE determines the pattern of the background. If SHADE is a texture, then the background is simply painted with it. If SHADE is a BITMAP, the background is tessellated (tiled) with it to cover the screen. If SHADE is T, it changes to the original shade, the value of WINDOWBACKGROUNDSHADE . It returns the previous value of the background.

### 19.6 SAVING BITMAPS

Bitmaps can be saved on files with the VARS file package command (page 11.22). The following two functions translate bitmaps into and out of a representation which may be used to transfer bitmaps between Interlisp and other computer systems' representations.

(READBITMAP) [Function]  
Creates a bitmap by reading an expression (written by PRINTBITMAP) from the primary input channel.

(PRINTBITMAP BITMAP ) [Function]  
Prints the bitmap BITMAP on the primary output channel in a format that can be read back in by READBITMAP.

### 19.7 SCREEN OPERATION

The following functions control the display screen.

## INTERLISP-D DISPLAY FACILITIES

(VIDEOCOLOR *BLACKFLAG*) [NoSpread Function]  
Sets the interpretation of the bits in the screen bitmap. If *BLACKFLAG* is NIL, a 0 bit will be displayed as white, otherwise a 0 bit will be displayed as black. VIDEOCOLOR returns the previous setting. If *BLACKFLAG* is not given, VIDEOCOLOR will return the current setting without changing anything.

Note: This function only works on the Xerox 1100 and Xerox 1108.

(VIDEORATE *TYPE*) [Function]  
Sets the rate at which the screen is refreshed. *TYPE* is one of NORMAL or TAPE. If *TYPE* is TAPE, the screen will be refreshed at the same rate as TV (60 cycles per second). This makes the picture look better when video taping the screen. Note: Changing the rate may change the dimensions of the display on the picture tube.

Several functions are provided for turning off the display (partially or completely). See page 18.22.

### 19.8 CHARACTERS AND FONTS

Fonts control the way characters look when printed on the screen or a graphics printer. Fonts are defined by a distinctive style or FAMILY (such as Gacha or TimesRoman), a SIZE (such as 10 points), and 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 the specification allows any combination, in practice the user will find that only certain combinations of families, sizes, faces, and rotations are available.

In specifying 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

A font also has the properties ASCENT, DESCENT, and HEIGHT (= ASCENT + DESCENT), and, for each character, a width and bit pattern. The ASCENT is the maximum height of any character in the font from its base line (the printing position). The DESCENT is the maximum extent of any character below the base line, such as the lower part of a "p." Therefore the top line of a character will be at Base+ASCENT-1, while the bottom line will be at Base-DESCENT. The width of each character specifies how a stream's position will change when the character is printed. This may have both an X and a Y component (e.g., for landscape fonts), and it varies from character to character in variable pitch fonts.

## Characters and Fonts

The information about a particular font is represented in a font descriptor. The following functions manipulate font descriptors:

(FONTCREATE FAMILY SIZE FACE ROTATION DEVICE NOERR ORFL G) [Function]  
Returns a font descriptor for the specified font. 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. For Interlisp-D, the possible values for DEVICE are DISPLAY for the display screen and PRESS for Press printers. DEVICE defaults to DISPLAY.

For display fonts, FONTCREATE looks for a STRIKE file with the appropriate name (such as TIMESROMAN8BI.STRIKE for a TIMESROMAN 8 BOLDITALIC font), searching through directories on the list FONTDIRECTORIES. If the file is found, it is read into a font descriptor. If the file is not found, FONTCREATE looks for fonts with less face information (in this example, TIMESROMAN8I.STRIKE) and fakes the remaining faces (such as by doubling the bit pattern of each character or slanting it). If no appropriately sized font is found, the action of the function is determined by NOERR ORFL G. If NOERR ORFL G is NIL, it generates a FILE NOT FOUND error with the name of the most specific file tried (in the example TIMESROMAN8BI.STRIKE); otherwise, FONTCREATE returns NIL.

For Press fonts, FONTCREATE accesses the widths information for the font from a font-dictionary file whose name is in the list FONTWIDTHSFILES (usually initialized in the site-greeting file to contain at least {DSK}FONTS.WIDTHS). That dictionary must contain information for the face as specified; there is no acceptable faking algorithm for hard-copy fonts. The width and height information for press fonts is expressed in micas (= 10 microns = 1/2540 inch), not in screen-point units.

The FAMILY argument to FONTCREATE may also be a list, in which case it is interpreted as a FAMILY-SIZE-FACE-ROTATION quadruple. 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.

(FONTP X) [Function]  
Returns X if X is a font descriptor; NIL otherwise.

The following functions take a font as one argument. This argument must either be a particular font descriptor or coerceable to a font descriptor. A display stream is coerced to its current font, a window is coerced to the current font of its display stream, and anything else is coerced by applying FONTCREATE to it.

(FONTPROP FONT PROP) [Function]  
Returns the value of the PROP property of font FONT. PROP may be one of FAMILY, SIZE, FACE, WEIGHT, SLOPE, EXPANSION, DEVICE, ASCENT, DESCENT, HEIGHT, or ROTATION.

(FONTCOPY OLDFONT PROP<sub>1</sub> VAL<sub>1</sub> PROP<sub>2</sub> VAL<sub>2</sub> ...) [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



## INTERLISP-D DISPLAY FACILITIES

and values. Thus, (FONTCOPY FONT 'WEIGHT 'BOLD 'DEVICE 'PRESS) will return a bold press font with all other properties the same as those of FONT. FONTCOPY accepts all the properties that FONTPROP interrogates except for ASCENT, DESCENT, and HEIGHT. If the rst property is a list, it is taken to be the PR OP<sub>1</sub> VAL<sub>1</sub> PR OP<sub>2</sub> VAL<sub>2</sub> sequence. Thus, (FONTCOPY FONT '(WEIGHT BOLD DEVICE PRESS)) is equivalent to the example above.

(CHARWIDTH CHAR CODE FONT ) [Function]  
CHAR CODE is an integer that represents a valid character (as returned by CHCON1). Returns the amount by which a stream's X-position will be incremented when the character is printed.

(CHARWIDTHY CHAR CODE FONT ) [Function]  
Like CHARWIDTH, but returns the Y component of the character's width, the amount by which a 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 PRIN2FL G 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 a display stream, its font is used. If PRIN2FL G is non-NIL, the PRIN2-pname of STR with respect to the readtable RDTBL is used.

(STRINGREGION STR WINDO W PRIN2FL G RDTBL ) [Function]  
Returns the region occupied by STR if it were printed at the current location in WINDO W. This is useful for determining where text is in a window to allow the user to select it. The arguments PRIN2FL G and RDTBL are passed to STRINGWIDTH.

It is sometimes useful to simulate an unavailable font or to use a font with characteristics different from the interpretations provided by the system. The following function allows the user to tell the system what font descriptor to use for given characteristics.

(SETFONTDESCRIPTOR FAMILY SIZE FACE ROTATION DEVICE FONT ) [Function]  
Indicates to the system that FONT is the font 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.

(DEFAULTFONT DEVICE FONT \_ ) [Function]  
Returns the font that would be used as the default (if NIL were specified as a font argument) for device DEVICE. If FONT is a font descriptor, it is set to be the default font for DEVICE.

The following functions allow the user to access and change the bitmaps for individual characters in a display font.

(GETCHARBITMAP CHAR CODE FONT ) [Function]  
Returns a bitmap containing a copy of the image of the character CHAR CODE in the font FONT.

## Display Streams

(PUTCHARBITMAP CHAR CODE FONT NEW CHARBITMAP ) [Function]  
Changes the bitmap image of the character CHAR CODE in the font FONT to the bitmap NEW CHARBITMAP . Currently, NEW CHARBITMAP must be the same width and height as the current image for CHAR CODE in the font FONT .

Users can interactively edit characters using the EDITCHAR function (page 20.10).

### 19.9 DISPLAY STREAMS

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. Display streams are a type of stream that also provides an interface for translation, clipping, and figure generation on bitmaps. All of the operations that can be applied to streams can be applied to display streams. For example, a display stream can be passed as the argument to PRINT, to print something on the bitmap of a display stream. In addition, special functions are provided to draw lines and curves and perform other graphical operations on display streams. Calling these functions on a stream that is not a display stream will generate an error.

Windows are closely related to display streams and can be thought of as a type of display stream. (In the near future, windows will be a type of display stream.) All of the functions that operate on display streams also accept windows.

Display streams can be created with the following function:

(DSPCREATE DESTINATION ) [Function]  
Returns a display stream, with initial settings as indicated below. If DESTINATION is specified, it is used as the destination bitmap, otherwise the screen bitmap is used.

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

(WFROMDS DISPLAYSTREAM ) [Function]  
Returns the window associated with DISPLAYSTREAM , creating a window if one does not exist. Returns NIL if the destination of DISPLAYSTREAM is not a screen bitmap that supports a window system.

#### 19.9.1 Manipulating Display Streams

The following functions manipulate the fields of a display stream (they may also be given a window, in which case the associated display stream is used). 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 in the display stream's destination bitmap; just the effect of future operations done through the display stream.

Warning: The window system maintains the Destination, XO set, YO set, and ClippingRegion fields of each window's display stream, adjusting them during window operations. Users should be very careful about changing these fields in a window's display stream (with DSPDESTINATION, DSPXOFFSET, DSPYOFFSET, or DSPCLIPPINGREGION).

## INTERLISP-D DISPLAY FACILITIES

- (DSPDESTINATION DESTINATION DISPLAYSTREAM ) [Function]  
Destination: The bitmap that the display stream modifies. This can be either the screen bitmap, or an auxiliary bitmap in order to construct figures, possibly save them, and then display them in a single operation. Initially the screen bitmap.
- (DSPXOFFSET XOFFSET DISPLAYSTREAM ) [Function]  
(DSPYOFFSET YOFFSET DISPLAYSTREAM ) [Function]  
XOffset: The X origin of the display stream's coordinate system in the destination bitmap's coordinate system. Initially 0 (no X-coordinate translation).  
YOffset: The Y origin of the display stream's coordinate system in the destination bitmap's coordinate system. Initially 0 (no Y-coordinate translation).  
Display streams have their own coordinate system. 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.
- (DSPCLIPPINGREGION REGION DISPLAYSTREAM ) [Function]  
ClippingRegion: A region that limits the extent of characters printed and lines drawn (in the display stream's coordinate system). Initially set so that no clipping occurs.
- (DSPXPOSITION XPOSITION DISPLAYSTREAM ) [Function]  
(DSPYPOSITION YPOSITION DISPLAYSTREAM ) [Function]  
XPosition: The current X position. Initially 0.  
YPosition: The current Y position. Initially 0.  
DSPXPOSITION and DSPYPOSITION specify the "current position" of the display stream, the position (in the display stream's coordinate system) where the next printing operation will start from. The functions which print characters or draw on a display stream update these values appropriately.
- (DSPTEXTURE TEXTURE DISPLAYSTREAM ) [Function]  
Texture: A texture that is the background pattern used for the display stream. Initially the value of WHITESHADE.
- (DSPFONT FONT DISPLAYSTREAM ) [Function]  
Font: A Font Descriptor that specifies the font used when printing characters to the display stream. Initially Gacha 10.  
Note: DSPFONT determines its new font descriptor from FONT by the same coercion rules that FONTPROP and FONTCOPY use, with one additional possibility: If FONT is a list of the form (PROP<sub>1</sub> VAL<sub>1</sub> PROP<sub>2</sub> VAL<sub>2</sub> ) where PROP<sub>1</sub> is acceptable as a font-property to FONTCOPY, then the new font is obtained by (FONTCOPY (DSPFONT NIL DISPLAYSTREAM) PROP<sub>1</sub> VAL<sub>1</sub> PROP<sub>2</sub> VAL<sub>2</sub> ).
- (DSPLEFTMARGIN XPOSITION DISPLAYSTREAM ) [Function]  
LeftMargin: An integer that is the X position after an end-of-line (in the display stream's coordinate system) - initially 0.
- (DSPRIGHTMARGIN XPOSITION DISPLAYSTREAM ) [Function]  
RightMargin: An integer that is the maximum X position that characters will

## Drawing on Windows and Display Streams

be printed at (in the display stream's coordinate system) - initially the value of SCREENWIDTH. This determines when an end of line is automatically inserted by the printing functions.

The line length of a window or display stream (as returned by LINELENGTH, page 6.8) 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, LeftMargin, or RightMargin are changed.

(DSPSOURCETYPE SOUR CETYPE DISPLAYSTREAM ) [Function]  
Source Type: The BITBLT sourcetype used when printing characters to the display stream. Must be either INPUT or INVERT. Initially INPUT.

(DSPOPERATION OPERA TION DISPLAYSTREAM ) [Function]  
Operation: The default BITBLT operation (REPLACE, PAINT, INVERT, or ERASE) used when printing or drawing on the display stream. Initially REPLACE.

(DSPLINEFEED DEL TAY DISPLAYSTREAM ) [Function]  
LineFeed: An integer that species the Y increment for each linefeed, normally negative. Initially minus the height of the initial font (Gacha 10).

(DSPSCROLL SWITCHSETTING DISPLAYSTREAM ) [Function]  
Scroll: A ag that determines the scrolling behavior of the display stream; either ON or OFF. If ON, the bits in the display streams's destination 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 XO set, YO set, or ClippingRegion elds. Initially OFF. (Note: if SWITCHSETTING is NIL, the Scroll eld is *not* changed, and the previous value is returned.)

### 19.9.2 Drawing on Windows and Display Streams

(DSPFILL REGION TEXTURE OPERA TION DISPLAYSTREAM ) [Function]  
Fills REGION of the destination bitmap (within the clipping region) with TEXTURE (a pattern of bits). If REGION is NIL, the whole destination (within the clipping region) is used. If TEXTURE or OPERA TION are NIL, the values from DISPLAYSTREAM are used.

(FILLCIRCLE X Y RADIUS TEXTURE DISPLAYSTREAM ) [Function]  
Fills in a circular area of radius RADIUS about the point (x,y) in the destination bitmap of DISPLAYSTREAM with TEXTURE . DISPLAYSTREAM 's position is left at (x,y).

(DSPRESET DISPLAYSTREAM ) [Function]  
Sets the X position of DISPLAYSTREAM to its left margin, sets its Y position to the top of the clipping region minus the font ascent, and lls its destination bitmap with its background Texture.

(MOVETO X Y DISPLAYSTREAM ) [Function]  
Changes the current position of DISPLAYSTREAM to the point (x,y).

(RELMOVETO DX DY DISPLAYSTREAM ) [Function]  
Changes the current position to the point (dx,dy) coordinates away from current

## INTERLISP-D DISPLAY FACILITIES

position of DISPLAYSTREAM .

(MOVETOUPPERLEFT DISPLAYSTREAM REGION ) [Function]  
 Changes the X position to the left edge of REGION and the Y position to the top of REGION less the font height of DISPLAYSTREAM . This is the beginning position of the top line of text in this region. If REGION is NIL, the clipping region of DISPLAYSTREAM is used. Note: this does not set the X position to the left margin like the function DSPRESET does.

(DSPBACKUP WIDTH DISPLAYSTREAM ) [Function]  
 Backs up DISPLAYSTREAM over a character which is WIDTH screen points wide. DSPBACKUP lls 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.

(CENTERPRINTINREGION EXP REGION DISPLAYSTREAM ) [Function]  
 Prints EXP so that it is centered within REGION of the DISPLAYSTREAM . If REGION is NIL, EXP will be centered in the clipping region of DISPLAYSTREAM .

### 19.9.3 Drawing Lines and Curves

Interlisp- D provides several functions for drawing lines and curves onto the destination bitmap of a display stream or window. The curve drawing functions take their BITBLT operation from the display stream, while for straight lines the Operation may be specified as an argument to the drawing function, with the display stream's operation only being used by default.

The following functions produce straight lines of the specified width (in screen points; the default is 1) in the display stream's destination bitmap. They do not allow "brush" patterns; however, they do support INVERT mode in which redrawing a line will erase it. These functions are intended for interactive applications where efficiency is important. DRAWCURVE can be used to draw lines with brushes.

(DRAWTO X Y WIDTH OPERATION DISPLAYSTREAM COLOR ) [Function]  
 Draws a line from the current position to the point (X,Y) onto the destination bitmap of DISPLAYSTREAM . The position of DISPLAYSTREAM is set to (X,Y).  
 If the destination bitmap has multiple bits per pixel, COLOR is a color specification that determines the color used to draw the line (See page 19.44). If COLOR is NIL, this will be the DSPCOLOR of DISPLAYSTREAM .

(RELDRAWTO DX DY WIDTH OPERATION DISPLAYSTREAM COLOR ) [Function]  
 Draws a line from the current position to the point (DX,DY) coordinates away onto the destination bitmap of DISPLAYSTREAM . The position of DISPLAYSTREAM is set to the end of the line.

(DRAWLINE X<sub>1</sub> Y<sub>1</sub> X<sub>2</sub> Y<sub>2</sub> WIDTH OPERATION DISPLAYSTREAM COLOR ) [Function]  
 Draws a line from the point (X<sub>1</sub>,Y<sub>1</sub>) to the point (X<sub>2</sub>,Y<sub>2</sub>) onto the destination bitmap of DISPLAYSTREAM . The position of DISPLAYSTREAM is set to (X<sub>2</sub>,Y<sub>2</sub>).

(DRAWBETWEEN POSITION<sub>1</sub> POSITION<sub>2</sub> WIDTH OPERATION DISPLAYSTREAM COLOR ) [Function]  
 Draws a line from the point POSITION<sub>1</sub> to the point POSITION<sub>2</sub> onto the destination

## Typescript Facilities: The “T” File

bitmap of `DISPLAYSTREAM` . The position of `DISPLAYSTREAM` is set to `POSITION 2`.

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 currently recognized shapes are `ROUND`, `SQUARE`, `HORIZONTAL`, `VERTICAL`, and `DIAGONAL`. A brush size is an integer specifying the width of the brush in screen points. The color is a color specification (see page 19.44), which is only used if the curve is drawn on a multiple bits per pixel bitmap.

A brush is specified to the various drawing functions as a shape-width-color list (such as `(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. Finally, if a brush is specified as `NIL`, a `(ROUND 1)` brush is used as default.

If a brush is a list atom, it is assumed to be a function which is called at each point of the curve’s trajectory with three arguments: the X-coordinate or the point, the Y-coordinate, and the display stream.

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 points indicated by the first element of the dashing list. The brush is *o*, not placed in the bitmap, for a 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. A curve or line is not dashed if the dashing argument to the drawing function is `NIL`.

The curve functions use the display stream’s clipping region and operation. Because of the problem of overlapping brush points, the `REPLACE` and `INVERT` operations are not implemented.

`(DRAWCURVE KNOTS CLOSED BRUSH DASHING DISPLAYSTREAM )` [Function]  
Draws a spline curve. `KNOTS` is a list of positions to which the spline will be fitted. `CLOSED` is a flag which indicates whether or not the spline is to be closed. The other arguments are interpreted as described above.

`(DRAWCIRCLE X Y RADIUS BRUSH DASHING DISPLAYSTREAM )` [Function]  
Draws a circle of radius `RADIUS` about the point `(X,Y)` onto the destination bitmap of `DISPLAYSTREAM` . `DISPLAYSTREAM` ’s position is left at `(X,Y)`. (Dashing may not be implemented for this function yet.) The other arguments are interpreted as described above.

`(DRAWELLIPSE X Y SEMIMINORRADIUS SEMIMAJORRADIUS ORIENTATION BRUSH DASHING DISPLAYSTREAM )` [Function]  
Draws an ellipse with a minor radius of `SEMIMINORRADIUS` and a major radius of `SEMIMAJORRADIUS` about the point `(X,Y)` onto the destination bitmap of `DISPLAYSTREAM` . `ORIENTATION` is the angle of the major axis in degrees, positive in the counterclockwise direction. `DISPLAYSTREAM` ’s position is left at `(X,Y)`. (Dashing may not be implemented for this function yet.) The other arguments are interpreted as described above.

### 19.10 TYPESCRIPT FACILITIES: THE “T” FILE

Output to the `le T` and echoing of type-in is directed to a distinguished terminal display stream. This is

## INTERLISP-D DISPLAY FACILITIES

initialized to be a display stream at the top of the screen, but that initial setting can be modified by the function `TTYDISPLAYSTREAM`.

(`TTYDISPLAYSTREAM` `DISPLAYSTREAM` ) [Function]  
Selects the display stream or window `DISPLAYSTREAM` to be the terminal output channel, and returns the previous terminal output display stream. `TTYDISPLAYSTREAM` puts `DISPLAYSTREAM` into scrolling mode and calls `PAGEHEIGHT` with the number of lines that will fit into `DISPLAYSTREAM` given its current Font and ClippingRegion. The `linelength` of `TTYDISPLAYSTREAM` is computed (like any other display stream) from its `LeftMargin`, `RightMargin`, and Font. If one of these fields is changed, its `linelength` is recalculated. If one of the fields used to compute the number of lines (such as the `ClippingRegion` or Font) changes, `PAGEHEIGHT` is not automatically recomputed. (`TTYDISPLAYSTREAM` (`TTYDISPLAYSTREAM`)) will cause it to be recomputed.

If the window system is active, the `line buffer` is saved in the old TTY window, and the `line buffer` is set to the one saved in the window of the new display stream, or to a newly created `line buffer` (if it does not have one). Caution: It is possible to move the `TTYDISPLAYSTREAM` to a nonvisible display stream or to a window whose current position is not in its clipping region.

(`CARET` `NEW CARET` ) [Function]  
Sets the shape that blinks at the location of the next output to the `TTYDISPLAYSTREAM`. `NEW CARET` is either (1) `NIL` - no changes, returns a `CURSOR` representing the current caret, (2) `OFF` - turns the caret off, or (3) a `CURSOR` which gives the new caret shape. The `hotspot` of `NEW CARET` indicates which point in the new caret bitmap should be located at the current output position. The previous caret is returned.

(`PAGEHEIGHT` `N` ) [Function]  
If `N` is greater than 0, it is the number of lines of output that will be printed to `TTYDISPLAYSTREAM` before the page is held. A page is held before the `N+1` line is printed to `TTYDISPLAYSTREAM` without intervening input if there is no terminal input waiting to be read. The output is held with the screen video reversed until a character is typed. Output holding is disabled if `N` is 0. `PAGEHEIGHT` returns the previous setting.

### 19.11 CURSOR AND MOUSE

The screen relative position at which the cursor bitmap is being displayed can be read or set using the functions:

(`CURSORPOSITION` `NEWPOSITION` `DISPLAYSTREAM` `OLDPOSITION` ) [Function]  
This returns the location of the cursor in the coordinate system of `DISPLAYSTREAM` (the current display stream, if `DISPLAYSTREAM` is `NIL`). If `OLDPOSITION` is a `POSITION`, it will be reused, and returned. If `NEWPOSITION` is non-`NIL`, it should be a `position` and the cursor will be positioned at `NEWPOSITION`.

## Mouse Button Testing

(ADJUSTCURSORPOSITION DEL TAX DEL TAY ) [Function]  
Moves the cursor DEL TAX points in the X direction and DEL TAY points in the Y direction. DEL TAX and DEL TAY default to 0.

The cursor can be changed like any other bitmap by BITBLTing into it or pointing a display stream at it and printing or drawing curves. For most applications, it is also necessary to locate the *hotspot* - a point within the CURSORWIDTH by CURSORHEIGHT area which is used to determine a *point* position for the cursor. Also for some applications it is necessary to save and restore the cursor. The Cursor record and the following functions provide these capabilities. A Cursor record has elds CURSORBITMAP and CURSORHOTSPOT, the latter a POSITION that gives the location of the hot spot inside the cursor.

(CURSORCREATE BITMAP X Y ) [Function]  
Returns a cursor object which has BITMAP as its image and the location (X,Y) as the hot spot. If x is a POSITION, it is used as the hot spot. If BITMAP has dimensions different from CURSORWIDTH by CURSORHEIGHT, the lesser of the widths and the lesser of the heights are used to determine the bits that actually get copied into the lower left corner of the cursor. If x is NIL, 0 is used. If y is NIL, CURSORHEIGHT-1 is used. The default cursor is an uparrow with its tip in the upper left corner and its hot spot at (0,CURSORHEIGHT-1).

(CURSOR NEW CURSOR \_ ) [Function]  
Returns a CURSOR record instance that contains (a copy of) the current cursor specification. If NEW CURSOR is a CURSOR record instance, the cursor will be set to the values in NEW CURSOR . If NEW CURSOR is T, the cursor will be set to the default cursor DEFAULTCURSOR, an upward left pointing arrow.

(SETCURSOR NEW CURSOR \_ ) [Function]  
If NEW CURSOR is a CURSOR record instance, the cursor will be set to the values in NEW CURSOR . This does not return the old cursor, and therefore, provides a way of changing the cursor without using storage.

(FLIPCURSOR) [Function]  
Inverts the cursor.

There are several cursors defined in Interlisp-D that may be of interest to users. One of these is WAITINGCURSOR, an hour glass shape used by the system to indicate that a long computation is in progress.

CURSORS can be saved on a file using the file package command CURSORS, or the UGLYVARS file package command.

### 19.11.1 Mouse Button Testing

There are various graphical input devices that can be read from Interlisp-D. The devices used in this manner are: a device called a mouse, which has three keys and steers the cursor, and seven uninterpreted keys on the keyboard. (Some Xerox 1100 systems may also have a small, *ve-* key keyset.) The following macros are provided to test the state of these input devices. (The three keys on the mouse (often called buttons) are referred to by their location: left, middle, or right.)

(MOUSESTATE BUTTONF ORM ) [Macro]  
Reads the mouse state and returns T if that state is described by BUTTONF ORM .



## INTERLISP-D DISPLAY FACILITIES

`BUTTONF ORM` can be one of the key indicators `LEFT`, `MIDDLE`, or `RIGHT`; the atom `UP` (indicating all keys are up); the form `(ONLY KEY )`; or a form of `AND`, `OR`, or `NOT` applied to any valid button form. For example: `(MOUSESTATE LEFT)` will be true if the left mouse button is down. `(MOUSESTATE (ONLY LEFT))` will be true if the left mouse button is the only one down. `(MOUSESTATE (OR (NOT LEFT) MIDDLE))` will be true if either the left mouse button is up or the middle mouse button is down.

`(LASTMOUSESTATE BUTTONF ORM )` [Macro]  
Similar to `MOUSESTATE`, but tests the value of `LASTMOUSEBUTTONS` rather than getting the current state. This is useful for determining which keys caused a `MOUSESTATE` to be true.

`(UNTILMOUSESTATE BUTTONF ORM INTER VAL )` [Macro]  
`BUTTONF ORM` is as described in `MOUSESTATE`. Waits until `BUTTONF ORM` is true or until `INTER VAL` milliseconds have elapsed. The value of `UNTILMOUSESTATE` is `T` if `BUTTONF ORM` was satisfied before it timed out, otherwise `NIL`. If `INTER VAL` is `NIL`, it waits indefinitely. It compiles into an open loop that calls the `TTY wait background` function. This form should not be used inside the `TTY wait background` function. `UNTILMOUSESTATE` does not use any storage during its wait loop.

The macros `KEYSETSTATE` and `LASTKEYSETSTATE` are identical to `MOUSESTATE` and `LASTMOUSESTATE` except that they also check the state of the `ve- nger` keyset as well as the state of the mouse buttons. That is they check the state of both the mouse and the keyset. Thus, if the left mouse button was the only mouse button held down, `(MOUSESTATE (ONLY LEFT))` would be `T` even though a keyset key was down; whereas `(KEYSETSTATE (ONLY LEFT))` would be `NIL` if a keyset button were down.

The names of the keyset keys are: `LEFTKEY`, `LEFTMIDDLEKEY`, `MIDDLEKEY`, `RIGHTMIDDLEKEY` and `RIGHTKEY`.

### 19.11.2 Low Level Access to Mouse

This section describes the low level access to the graphical input devices and can be skipped by most users. Graphical input information is represented in the following global variables:

`LASTMOUSEX` [Variable]  
The X position of the cursor in absolute screen coordinates. Also see the function `LASTMOUSEX` below.

`LASTMOUSEY` [Variable]  
The Y position of the cursor in absolute screen coordinates. Also see the function `LASTMOUSEY` below.

`LASTMOUSEBUTTONS` [Variable]  
An 8-bit number that has bits on corresponding to the mouse buttons that are down: 4Q is the left mouse button, 2Q is the right button, 1Q is the middle button. (Bits 200Q, 100Q, 40Q, 20Q, and 10Q give the state of the keyset keys, from left to right, if you have a keyset.)

## Windows

`LASTKEYBOARD` [Variable]  
The state of certain keys on the keyboard (200Q = lock, 100Q = left shift, 40Q = ctrl, 10Q = right shift, 4Q = blankBottom, 2Q = blankMiddle, 1Q = blankTop). If the key is down, the corresponding bit is on.

`LASTMOUSETIME` [Variable]  
The time in milliseconds since the mouse was last read (since the last call to `GETMOUSESTATE`. `LASTMOUSETIME` is a 16-bit positive integer so it rolls over every 65+ seconds.

The following functions provide low level cursor access in display stream coordinates.

`(LASTMOUSEX DISPLAYSTREAM )` [Function]  
Returns the value of the cursor's X position in the coordinates of `DISPLAYSTREAM`.

`(LASTMOUSEY DISPLAYSTREAM )` [Function]  
Returns the value of the cursor's Y position in the coordinates of `DISPLAYSTREAM`.

`(DECODEBUTTONS BUTTONSTATE )` [Function]  
Returns a list of the buttons or keys that are down in the state `BUTTONSTATE`. If `BUTTONSTATE` is not a `SMALLP`, `LASTMOUSEBUTTONS` is used (see `GETMOUSESTATE` below). The button names that can be returned are: `LEFT`, `MIDDLE`, `RIGHT` (the three mouse keys), `LEFTKEY`, `LEFTMIDDLEKEY`, `MIDDLEKEY`, `RIGHTMIDDLEKEY` and `RIGHTKEY` (the ve keyset keys).

`(GETMOUSESTATE)` [Function]  
Reads the current state of the mouse and sets the variables `LASTMOUSEX`, `LASTMOUSEY`, `LASTMOUSEBUTTONS`, `LASTMOUSETIME`, and `LASTKEYBOARD`. In polling mode, the program must remember the previous state and look for changes, such as a key going up or down, or the cursor moving outside a region of interest.

## 19.12 WINDOWS

Windows provide a means by which different programs can share the display harmoniously. Interlisp-D provides both interactive and programmatic constructs for creating, moving, reshaping, overlapping, and destroying windows in such a way that a program can be embedded in a window in a relatively transparent fashion. This is implemented by having each window save the bits that it obscures. This allows existing Interlisp programs to be used without change, while providing a base for experimentation with more complex window semantics in new applications.

Because the window system assumes that all programs follow certain conventions concerning control of the screen, ordinary user programs should not perform display operations directly on the screen. In particular, functions that can operate directly on bitmaps (such as `BITBLT` or `BITMAPBIT`) should not be given `(SCREENBITMAP)` as the destination argument. All interactions with the screen should take place through windows.

For specialized applications that require taking complete control of the display, the window system can be turned off (and back on again) with the following function:

## INTERLISP-D DISPLAY FACILITIES

(WINDOWWORLD FLAG)

[NoSpread Function]

The window world is turned on if FLAG is T and off if FLAG is NIL. WINDOWWORLD returns the previous state of the window world (T or NIL). If WINDOWWORLD is given no arguments, it simply returns the current state without affecting the window world.

### 19.12.1 What are Windows?

A window specifies a region of the screen, a display stream, a location in an occlusion stack, functions that get called when the window undergoes certain actions, and various other items of information. The basic model is that a window is a passive collection of bits (on the screen). On top of this basic level, the system supports many different types of windows that are linked to the data structures displayed in them and provide selection and redisplaying routines. In addition, it is possible for the user to create new types of windows by providing selection and displaying functions for them.

Windows are ordered in depth from user to background. Windows in front of others obscure the latter. Operating on a window generally brings it to the top.

Windows are located at a certain position on the screen. Each window has a clipping region that confines all bits splashed at it to a region that allows a border around the window, and a title above it.

Each window has a display stream associated with it, and either a window or its display stream can be passed interchangeably to all system functions. There are dependencies between the window and its display stream that the user should not disturb. For instance, the destination bitmap of the display stream of a window must always be (SCREENBITMAP). The XO set, YO set, and ClippingRegion attributes of the display stream should not be changed. At some future date, the notions of window and display stream will be merged.

Windows can be created by the user interactively, under program control, or may be created automatically by the system.

Windows are in one of two states: “open” or “closed”. In an “open” state, a window is on the occlusion stack and therefore visible on the screen (unless it is covered by other open windows) and accessible to mouse operations. In a “closed” state, a window is not on the occlusion stack and therefore not visible and not accessible to mouse operations. Any attempt to print or draw on a closed window will open it.

When Interlisp-D starts up, there are three windows on the screen: a top level typescript window, a window containing the Interlisp-D logo, and a prompt window. The top level typescript window corresponds to the `le T` in the EXEC process where the read-eval-print loop is operating. The logo window is bound to the variable LOGOW until it is closed. The prompt window is used for the printing of help or prompting messages. It is available to user programs through the following functions:

PROMPTWINDOW

[Variable]

Global variable containing the prompt window.

(PROMPTPRINT EXP)

[NoSpread Function]

Prints EXP in the prompt window.

(CLRSPROMPT)

[Function]

Clears the prompt window.

## Interactive Window Operations

### 19.12.2 Interactive Window Operations

The Interlisp-D window system allows the user to interactively manipulate the windows on the screen, moving them around, changing their shape, etc. by selecting various operations from a menu. Programmatic versions of these operations are described on page 19.26.

For most windows, depressing the RIGHT mouse key when the cursor is inside a window during I/O wait will cause the window to come to the top and a menu of window operations to appear. If a command is selected from this menu (by releasing the right mouse key while the cursor is over a command), the selected operation will be applied to the window in which the menu was brought up. (It is possible for an applications program to redefine the action of the RIGHT mouse key. In these cases, there is a convention that the default command menu may be brought up by depressing the RIGHT key when the cursor is in the header or border of a window. See page 19.30) The operations are:

CLEAR	[Window Menu Command] Clears the window and repositions it to the left margin of the first line of text (below the upper left corner of the window by the amount of the font ascent).
CLOSE	[Window Menu Command] Closes the window, i.e., removes it from the screen. (See CLOSEW, page 19.26.)
BURY	[Window Menu Command] Puts the window on the bottom of the occlusion stack, thereby exposing any windows that it was hiding.
MOVE	[Window Menu Command] Moves the window to a location specified by depressing and then releasing the LEFT key. During this time a ghost frame will indicate where the window will reappear when the key is released. (See GETBOXPOSITION, page 19.36.)
SHAPE	[Window Menu Command] Allows the user to specify a new region for the existing window contents. If the LEFT key is used to specify the new region, the reshaped window can be placed anywhere. If the MIDDLE key is used, the cursor will start out tugging at the nearest corner of the existing window, which is useful for making small adjustments in a window that is already positioned correctly.
REDISPLAY	[Window Menu Command] Redisplays the window. (See REDISPLAYW, page 19.27.)
PAINT	[Window Menu Command] Switches to a mode in which the cursor can be used like a paint brush to draw in a window. This is useful for making notes on a window. While the LEFT key is down, bits are added. While the MIDDLE key is down, they are erased. The RIGHT button pops up a command menu that allows changing of the brush shape, size and shade, changing the mode of combining the brush with the existing bits, or stopping paint mode.  Paint mode also contains a hardcopy command that makes a Press le of the bits in a window and sends it to the printer. There are limitations on the complexity and size of the bitmaps that some printers will print. If the printer does not print

## INTERLISP-D DISPLAY FACILITIES

the entire window correctly, try a smaller window or one with fewer black bits in it. To get a hardcopy of an arbitrary part of the screen that crosses window boundaries, use the `HARDCOPY` command in the background menu (below).

SNAP

[Window Menu Command]

Prompts for a region on the screen and makes a new window whose bits are a snapshot of the bits currently in that region. Useful for saving some particularly choice image before the window image changes.

Occasionally, a user will have a number of large windows on the screen, making it difficult to access those windows being used. To help with the problem of screen space management, the Interlisp-D window system allows the creation of *Icons*. An icon is a small rectangle (containing text or a bitmap) which is a “shrunk-down” form of a particular window. Using the `SHRINK` and `EXPAND` commands, the user can shrink windows not currently being used into icons, and quickly restore the original windows at any time.

SHRINK

[Window Menu Command]

Removes the window from the screen and brings up its icon. (See `SHRINKW`, page 19.27.) The window can be restored by selecting `EXPAND` from the window command menu of the icon.

If the `RIGHT` button is pressed while the cursor is in an icon, the window command menu will contain a slightly different set of commands. The `REDISPLAY` and `CLEAR` commands are removed, and the `SHRINK` command is replaced with the `EXPAND` command:

EXPAND

[Window Menu Command]

Restores the window associated with this icon and removes the icon. (See `EXPANDW`, page 19.28.)

If the `RIGHT` button is pressed while the cursor is not in any window, a “background menu” appears with the following operations:

SAVEVM

[Window Menu Command]

Calls the function `SAVEVM` (page 18.4), which writes out all of the dirty pages of the virtual memory. After a `SAVEVM`, and until the pagefault handler is next forced to write out a dirty page, your virtual memory image will be continuable (as of the `SAVEVM`) should you experience a system crash or other disaster.

SNAP

[Window Menu Command]

The same as the `SNAP` command described above.

HARDCOPY

[Window Menu Command]

Prompts for a region on the screen, makes a press le and sends it to the printer.

The printing is done with `HARDCOPYW` (page 18.18), so if `FULLPRESSPRINTER` is non-NIL, the image will be sent there, rather than to (`PRINTINGHOST`).

Some built-in facilities and `Lispusers` packages add commands to the background menu, to provide an easy way of calling the different facilities. The user can determine what these new commands do by holding the `RIGHT` button down for a few seconds over the item in question; an explanatory message will be printed in the prompt window.

## Changing Entries on the Window Command Menus

The following functions provide a functional interface to the interactive window operations so that user programs can call them directly.

(DOWINDOWCOM WINDO W ) [Function]  
If WINDO W is NIL, it calls DOBACKGROUNDCOM. If WINDO W is a shrunken window, it brings up the “icon window” menu. If WINDO W is a unshrunk window, it brings up the window menu. The initial items in these menus are described above. If the user selects one of the items from the provided menu, that item is APPLIED to WINDO W. If WINDO W is not a WINDOW or NIL, it returns.

(DOBACKGROUNDCOM) [Function]  
Brings up the background menu. The initial items in this menu are described above. If the user selects one of the items from the menu, that item is EVALed.

### 19.12.3 Changing Entries on the Window Command Menus

The window command menus for unshrunk windows, shrunken windows, and the background are cached in the variables WindowMenu, IconWindowMenu, and BackgroundMenu. To change the entries in these menus, the user should change the menu “command lists” in the variables WindowMenuCommands, IconWindowMenuCommands, and BackgroundMenuCommands, and set the appropriate menu variable to a non-MENU, so the menu will be recreated. This provides a way of adding commands to the menu, of changing its font or of restoring the menu if it gets clobbered. The “command lists” are in the format of the ITEMS eld of a menu (see page 19.39), except as specified below.

Note: command menus are recreated using the current value of MENUFONT.

WindowMenu [Variable]  
WindowMenuCommands [Variable]

The menu that is brought up in response to a right button in an unshrunk window is stored on the variable WindowMenu. If WindowMenu is set to a non-MENU, the menu will be recreated from the list of commands WindowMenuCommands. The CADR of each command added to WindowMenuCommands should be a function name that will be APPLIED to the window.

IconWindowMenu [Variable]  
IconWindowMenuCommands [Variable]

The menu that is brought up in response to a right button in a shrunken window is stored on the variable IconWindowMenu. If it is NIL, it is recreated from the list of commands IconWindowMenuCommands. The CADR of each command added to IconWindowMenuCommands should be a function name that will be APPLIED to the window.

BackgroundMenu [Variable]  
BackgroundMenuCommands [Variable]

The menu that is brought up in response to a right button in the background is stored on the variable BackgroundMenu. If it is NIL, it is recreated from the list of commands BackgroundMenuCommands. The CADR of each command added to BackgroundMenuCommands should be a form that will be EVALed.

## INTERLISP-D DISPLAY FACILITIES

### 19.12.4 Coordinate Systems

One way of thinking of a window is as a “view” onto an object (e.g. a graph, a file, a picture, etc.) The object has its own natural coordinate system in terms of which its subparts are laid out. When the window is created, the `XO` set and `YO` set of the window’s display stream are set to map the origin of the object’s coordinate system into the lower left point of the window’s interior region. At the same time, the `ClippingRegion` of the display stream is set to correspond to the interior of the window. From then on, the display stream’s coordinate system is translated and its clipping region adjusted whenever the window is moved, scrolled or reshaped.

There are several distinct regions associated with a window viewing an object. First, there is a region in the window’s coordinate system that contains the complete image of the object. This region (which can only be determined by application programs with knowledge of the “semantics” of the object) is stored as the `EXTENT` property of the window (page 19.32). Second, the clipping region of the window (obtainable with the function `DSPCLIPPINGREGION`) specifies the portion of the object that is actually visible in the window. This is set so that it corresponds to the interior of the window (not including the border or title). Finally, there is the region on the screen that specifies the total area that the window occupies, including the border and title. This region (in screen coordinates) is stored as the `REGION` property of the window (page 19.33).

### 19.12.5 Scrolling

The window system supports the idea of scrolling the contents of a window. Scrolling regions are on the left and the bottom edge of each window. The scrolling regions will only be active if the window has a `SCROLLFN` window property (page 19.31). If a window has a `SCROLLFN` and the cursor moves from inside that window into its scrolling region and remains there for `SCROLLWAITTIME` milliseconds (initially 1000), a scroll bar appears. The value of the global variable `SCROLLBARWIDTH` (initially 24) determines the size of the scrolling region. The `LEFT` key is used to indicate upward or leftward scrolling by the amount necessary to move the selected position to the top or the left edge. The `RIGHT` key is used to indicate downward or rightward scrolling by the amount necessary to move the top or left edge to the selected position. The `MIDDLE` key is used to indicate global placement of the object within the window (similar to “thumbing” a book).

In the scroll region, the part of the object that is being viewed by the window is marked with a gray shade. If the whole scroll bar is thought of as the entire object, the shaded portion is the portion currently being viewed. This will only occur when the window “knows” how big the object is (see window property `EXTENT`, page 19.32).

When the button is released in a scroll region, the function `SCROLLW` is called. `SCROLLW` calls the scrolling function associated with the window to do the actual scrolling and provides a programmable entry to the scrolling operation.

```
(SCROLLW WINDO W DEL TAX DEL TAY CONTINUOUSFL G) [Function]
Calls the SCROLLFN window property of the window WINDO W with argu-
ments WINDO W, DEL TAX, DEL TAY and CONTINUOUSFL G. See SCROLLFN window
property, page 19.31.
```

The function that tracks the mouse while it is in the scroll region is `SCROLL.HANDLER`.

## Scrolling

(SCROLL.HANDLER WINDO W )

[Function]

This is called when the cursor leaves a window in either the left or downward direction. If WINDO W does not have a scroll region for this direction (e.g. the window has moved or reshaped since it was last scrolled), a scroll region is created that is SCROLLBARWIDTH wide. It then waits for SCROLLWAITTIME milliseconds and if the cursor is still inside the scroll region, it opens a window the size of the scroll region and changes the cursor to indicate the scrolling is taking place.

When a button is pressed, the cursor shape is changed to indicate the type of scrolling (up, down, left, right or thumb). After the button is held for WAITBEFORESCROLLTIME milliseconds, until the button is released SCROLLW is called each WAITBETWEENSCROLLTIME milliseconds. These calls are made with the CONTINUOUSFL G argument set to T. If the button is released before WAITBEFORESCROLLTIME milliseconds, SCROLLW is called with the CONTINUOUSFL G argument set to NIL.

The arguments passed to SCROLLW depend on the mouse button. If the LEFT button is used in the vertical scroll region, DY is distance from cursor position at the time the button was released to the top of the window and DX is 0. If the RIGHT button is used, the inverse of this quantity is used for DY and 0 for DX. If the LEFT button is used in the horizontal scroll region, DX is distance from cursor position to left of the window and DY is 0. If the RIGHT button is used, the inverse of this quantity is used for DX and 0 for DY.

If the MIDDLE button is pressed, the distance argument to SCROLLW will be a FLOATP between 0.0 and 1.0 that indicates the proportion of the distance the cursor was from the left or top edge to the right or bottom edge.

SCROLLBYREPAINTFN is the standard scrolling function which should be used as the SCROLLFN property for most scrolling windows.

(SCROLLBYREPAINTFN WINDO W DEL TAX DEL TAY CONTINUOUSFL G )

[Function]

This function, when used as a SCROLLFN, BITBLTs the bits that will remain visible after the scroll to their new location, fills the newly exposed area with texture, adjusts the window's coordinates and then calls the window's REPAINTFN on the newly exposed region. Thus this function will scroll any window that has a repaint function. If WINDO W has an EXTENT property (page 19.32), SCROLLBYREPAINTFN will limit scrolling to keep the extent region visible or near visible. That is, it will not scroll the window so that the top of the extent is below the top of the window, the bottom of the extent is more than one point above the top of the window, the left of the extent is to the right of the window and the right of the extent is to the left of the window. The EXTENT is scrolled to just above the window to provide a way of "hiding" the contents of a window.

If DEL TAX or DEL TAY is a FLOATP, SCROLLBYREPAINTFN will position the window so that its top or left edge will be positioned at that proportion of its EXTENT. If the window does not have an EXTENT, SCROLLBYREPAINTFN will do nothing.

If CONTINUOUSFL G is non-NIL, this indicates that the scrolling button is being held down. In this case, SCROLLBYREPAINTFN will scroll the distance of one linefeed height (as returned by DSPLINEFEED, page 19.12).



## INTERLISP-D DISPLAY FACILITIES

### 19.12.6 Programmatic Window Operations

- (CREATEW REGION TITLE BORDER NOOPENFL G) [Function]  
Creates a new window. REGION indicates where and how large the window should be by specifying the exterior region of the window (the usable height and width of the resulting window will be smaller than the height and width of the region by twice the border size and further less the height of the title, if any). If REGION is NIL, GETREGION is called to prompt the user for a region.
- If TITLE is non-NIL, it is printed in the border at the top of the window. The TITLE is printed using the global display stream WindowTitleDisplayStream. Thus the height of the title will be (FONTPROP WindowTitleDisplayStream 'HEIGHT).
- If BORDER is a number, it is used as the border size. If BORDER is not a number, the window will have a border WBorder (initially 4) bits wide.
- If NOOPENFL G is non-NIL, the window will not be opened, i.e. displayed on the screen.
- (WINDOWP X) [Function]  
Returns X if X is a window, NIL otherwise.
- (OPENWP WINDO W) [Function]  
Returns WINDO W, if WINDO W is an open window (has not been closed); NIL otherwise.
- (OPENWINDOWS) [Function]  
Returns a list of all active windows.
- (WHICHW X Y) [Function]  
Returns the window which contains the position in screen coordinates of X if X is a POSITION, the position (X,Y) if X and Y are numbers, or the position of the cursor if X is NIL. Returns NIL if the coordinates are not in any window. If they are in more than one window, it returns the uppermost.
- Example: (WHICHW) returns the window that the cursor is in.
- (DECODE/WINDOW/OR/DISPLAYSTREAM DSOR W WINDO WV AR TITLE BORDER) [Function]  
If DSOR W is a display stream, it is returned. If DSOR W is a window, its display stream is returned. If DSOR W is NIL, it evaluates WINDO WV AR (which should be an atom). If its value is a window, it is reopened if it is closed, and returned. If its value is not a window, WINDO WV AR is set to a newly created window (prompting user for region) and returned. If DSOR W is NEW, a new window is created and returned. If TITLE or BORDER are given and a window is involved, the TITLE or BORDER property of the window is reset. The DSOR W = NIL case is most useful for programs that want to display their output in a window, but want to reuse the same window each time they are called. The non-NIL cases are good for decoding a display stream argument passed to a function.
- (WIDTHIFWINDOW INTERIOR WIDTH BORDER) [Function]  
Returns the width of the window necessary to have INTERIOR WIDTH points in its

## Programmatic Window Operations

interior if the width of the border is `BORDER` . If `BORDER` is `NIL`, the default border size `WBorder` is used.

(`HEIGHTIFWINDOW` `INTERIORHEIGHT` `TITLEFLG` `BORDER` ) [Function]  
Returns the height of the window necessary to have `INTERIORHEIGHT` points in its interior with a border of `BORDER` and, if `TITLEFLG` is non-`NIL`, a title. If `BORDER` is `NIL`, the default border size `WBorder` is used.

`WIDTHIFWINDOW` and `HEIGHTIFWINDOW` are useful for calculating the width and height for a call to `GETBOXPOSITION` for the purpose of positioning a window.

Interlisp-D provides a set of operations which apply to any window. In addition to being available as functions, most of these are also available via the standard mouse interface. See page 19.20

(`TOTOPW` `WINDOW` `W` `NOCALL` `TOPWFN` ) [Function]  
Brings `WINDOW` `W` to the top of the stack of overlapping windows, guaranteeing that it is entirely visible. If `WINDOW` `W` is closed, it is opened. This is done automatically whenever a printing or drawing operation occurs to the window.

If `NOCALL` `TOPWFN` is `NIL`, the `TOTOPFN` of `WINDOW` `W` is called (page 19.30). If `NOCALL` `TOPWFN` is `T`, it is not called, which allows a `TOTOPFN` to call `TOTOPW` without causing an infinite loop.

(`SHAPEW` `WINDOW` `W` `NEWREGION` ) [Function]  
Reshapes `WINDOW` `W` to the region `NEWREGION` , or prompts for a region (with `GETREGION`, page 19.37) if none is supplied. Calls the window's `RESHAPEFN`, if any (page 19.31).

(`CLOSEW` `WINDOW` `W` ) [Function]  
`CLOSEW` calls the function or functions on the window property `CLOSEFN` of `WINDOW` `W` , if any (page 19.30). If one of the `CLOSEFN`s is the atom `DON'T` or returns the atom `DON'T` as a value, `CLOSEW` returns without doing anything further. Otherwise, `CLOSEW` removes `WINDOW` `W` from the window stack and restores the bits it is obscuring. If `WINDOW` `W` was closed, `WINDOW` `W` is returned as the value. If it was not closed, (for example because its `CLOSEFN` returned the atom `DON'T`), `NIL` is returned as the value.

`WINDOW` `W` can be restored in the same place with the same contents (reopened) by calling `OPENW` or by using it as the source of a display operation.

(`OPENW` `WINDOW` `W` ) [Function]  
If `WINDOW` `W` is a closed window, `OPENW` calls the function or functions on the window property `OPENFN` of `WINDOW` `W` , if any (page 19.30). If one of the `OPENFN`s is the atom `DON'T`, the window will not be opened. Otherwise the window is placed on the occlusion stack of windows and its contents displayed on the screen. If `WINDOW` `W` is an open window, it returns `NIL`.

(`MOVEW` `WINDOW` `W` `POSorX` `Y` ) [Function]  
Moves `WINDOW` `W` to the position specified by `POSorX` and `Y` according to the following rules:

If `POSorX` is `NIL`, `GETBOXPOSITION` (page 19.36) is called to read a position from

## INTERLISP-D DISPLAY FACILITIES

the user.

If `POSorX` is a `POSITION`, `POSorX` is used.

If `POSorX` and `Y` are both `NUMBERP`, a position is created using `POSorX` as the `XCOORD` and `Y` as the `YCOORD`.

If `POSorX` is a `REGION`, a position is created using its `LEFT` as the `XCOORD` and `BOTTOM` as the `YCOORD`.

If `WINDO W` is not open and `POSorX` is non-`NIL`, the window will be moved without being opened. Otherwise, it will be opened.

If `WINDO W` has the atom `DON'T` as a `MOVEFN` property (page 19.32), the window will not be moved. If `WINDO W` has any other non-`NIL` value as a `MOVEFN` property, it should be a function or list of functions that will be called before the window is moved with the `WINDO W` as an argument. If it returns the atom `DON'T`, the window will not be moved. If it returns a position, the window will be moved to that position instead of the one specified by `POSorX` and `Y`. If there are more than one `MOVEFN`s, the last one to return a value is the one that determines where the window is moved to.

If `WINDO W` is moved and `WINDO W` has a window property of `AFTERMOVEFN` (page 19.32), it should be a function or a list of functions that will be called after the window is moved with `WINDO W` as an argument.

`MOVEW` returns the new position, or `NIL` if the window could not be moved.

(`RELMOVEW WINDO W POSITION`) [Function]  
Like `MOVEW` for moving windows but `POSITION` is interpreted relative to the current position of `WINDO W`. Example: The following code moves `WINDO W` to the right one screen point.

(`RELMOVEW WINDO W (create POSITION XCOORD _ 1 YCOORD _ 0)`)

(`CLEARW WINDO W`) [Function]  
Fills `WINDO W` with its background texture, changes its coordinate system so that the origin is the lower left corner of the window, sets its `X` position to the left margin and sets its `Y` position to the base line of the uppermost line of text, ie. the top of the window less the font ascent.

(`BURYW WINDO W`) [Function]  
Puts `WINDO W` on the bottom of the stack by moving all the windows that it covers in front of it.

(`REDISPLAYW WINDO W REGION ALWAYSFLG`) [Function]  
Redisplay the region `REGION` of the window `WINDO W`. If `REGION` is `NIL`, the entire window is redisplayed. If `ALWAYSFLG` is `NIL`, and `WINDO W` doesn't have a `REPAINTFN` (page 19.32), `WINDO W` will not change and the message "That window doesn't have a `REPAINTFN`" will be printed in the prompt window.

(`SHRINKW WINDO W TO WHAT ICONPOSITION EXPANDFN`) [Function]  
`SHRINKW` makes a small icon which represents `WINDO W` and removes `WINDO W`

## Window Properties

from the screen. Icons have a different window command menu that contains ‘EXPAND’ instead of ‘SHRINK’. The EXPAND command calls EXPANDW which returns the shrunken window to its original size and place.

The SHRINKFN property (page 19.30) of the window WINDO W affects the operation of SHRINKW. If the SHRINKFN property of WINDO W is the atom DON’T, SHRINKW prints “Can’t shrink that window” in the PROMPTWINDOW and returns. Otherwise, the SHRINKFN property of the window is treated as a (list of) function(s) to apply to WINDO W ; if any returns the atom DON’T, SHRINKW prints “Can’t shrink that window” in the PROMPTWINDOW and returns.

TO WHA T, if given, indicates the image the icon window will have. If TO WHA T is a string, atom or list, the icon’s image will be that string (currently implemented as a title-only window with TO WHA T as the title.) If TO WHA T is a BITMAP, the icon’s image will be a copy of the bitmap. If TO WHA T is a WINDOW, that window will be used as the icon.

If TO WHA T is not given (as is the case when invoked from the SHRINK window command), then the following apply in turn: (1) If the window has an ICONFN property (page 19.31), it gets called with arguments (WINDO W OLDICON ), where WINDO W is the window being shrunk and OLDICON is the previously created icon, if any. The ICONFN should return one of the TO WHA T entities described above or return the OLDICON if it does not want to change it. (2) If the window has an ICON property (page 19.31), it is used as the value of TO WHA T. (3) If the window has neither an ICONFN or ICON property, the icon will be WINDO W ’s title or, if WINDO W doesn’t have a title, the date and time of the icon creation.

ICONPOSITION gives the position that the new icon will be on the screen. If it is NIL, the icon will be in the corner of the window furthest from the center of the screen.

In all cases the icon is cached on the property ICONWINDOW (page 19.31) of WINDO W so repeating SHRINKW reuses the same icon (unless overridden by the ICONFN described above). Thus to change the icon it is necessary to remove the ICONWINDOW property or call SHRINKW explicitly giving a TO WHA T argument.

(EXPANDW ICON )

[Function]

Restores the window for which ICON is an icon, and removes the icon from the screen. If the EXPANDFN (page 19.31) window property of the main window is the atom DON’T, the window won’t be expanded. Otherwise, the window will be restored to its original size and location and the EXPANDFN (or list of functions) will be applied to it.

### 19.12.7 Window Properties

The behavior of a window is controlled by a set of window properties. Some of these are used by the system. However, any arbitrary property name may be used by a user program to associate information with a window. For many applications the user will associate the structure being displayed with its window using a property. The following functions provide for reading and setting window properties:

## INTERLISP-D DISPLAY FACILITIES

(WINDOWPROP WINDO W PR OP NEWV ALUE ) [NoSpread Function]  
Returns the previous value of WINDO W 's PR OP aspect. If NEWV ALUE is given, (even if given as NIL), it is stored as the new PR OP aspect. Some aspects cannot be set by the user and will generate errors. Any PR OP name that is not recognized is stored on a property list associated with the window.

(WINDOWADDPROP WINDO W PR OP ITEMTO ADD ) [Function]  
WINDOWADDPROP adds a new item to a window property. If ITEMTO ADD is EQ to an element of the PR OP property of the window WINDO W , nothing is added. If the current property is not a list, it is made a list before ITEMTO ADD added. WINDOWADDPROP returns the previous property. The new item always goes on the end of the list. (Note: If the order of items in the list is important, the list can be modified using WINDOWPROP.) WINDOWADDPROP is useful for adding OPENFN or CLOSEFN functions to a window without affecting its existing functions.

(WINDOWDELPROP WINDO W PR OP ITEMTODELETE ) [Function]  
WINDOWDELPROP deletes ITEMTODELETE from the window property PR OP of WINDO W and returns the previous list if ITEMTODELETE was an element. If ITEMTODELETE was not a member of window property PR OP , NIL is returned.

### 19.12.7.1 Mouse Function Window Properties

These properties allow the user to control the response to mouse activity in a window. The value of these properties, if non-NIL, should be a function that will be called (with the window as argument) when the specified event occurs.

Note: these functions should be “self-contained”, communicating with the outside world solely via their window argument, e.g., by setting window properties. In particular, these functions should not expect to access variables bound on the stack, as the stack context is formally undefined at the time these functions are called. Since the functions are invoked asynchronously, they perform any TTY input operations from their own window.

WINDOWENTRYFN [Window Property]  
Whenever a button goes down in the window and the process associated with the window (stored under the PROCESS property) is not the tty process, the WINDOWENTRYFN is called. The default is GIVE.TTY.PROCESS (page 18.34) which gives the process associated with the window the tty and calls the BUTTONEVENTFN.

CURSORINFN [Window Property]  
Whenever the mouse moves into the window, the CURSORINFN is called.

CURSOROUTFN [Window Property]  
The CURSOROUTFN is called when the cursor leaves the window.

CURSORMOVEDFN [Window Property]  
The CURSORMOVEDFN is called whenever the cursor has moved and is inside the window. This allows a window function to implement “active” regions within itself by having its CURSORMOVEDFN determine if the cursor is in a region of interest, and if so, perform some action.

## Event Window Properties

BUTTONEVENTFN

[Window Property]

The BUTTONEVENTFN is called whenever there is a change in the state (up or down) of the mouse buttons inside the window. Changes to the mouse state while the BUTTONEVENTFN is running will not be interpreted as new button events, and the BUTTONEVENTFN will not be re-invoked.

RIGHTBUTTONFN

[Window Property]

The RIGHTBUTTONFN is called in lieu of the standard window menu operation (DOWINDOWCOM) when the RIGHT key is depressed in a window. More specifically, the RIGHTBUTTONFN is called instead of the BUTTONEVENTFN when (MOUSESTATE (ONLY RIGHT)). If the RIGHT key is to be treated like any other key in a window, supply RIGHTBUTTONFN and BUTTONEVENTFN with the same function.

Note: When an application program defines its own RIGHTBUTTONFN, there is a convention that the default RIGHTBUTTONFN, DOWINDOWCOM (page 19.22), may be executed by depressing the RIGHT key when the cursor is in the header or border of a window. User programs are encouraged to follow this convention.

### 19.12.7.2 Event Window Properties

CLOSEFN

[Window Property]

The CLOSEFN window property can be a single function or a list of functions that are called just before a window is closed by CLOSEW (page 19.26). (Note: If the CAR of the list is a LAMBDA word, it is treated as a single function.) The function(s) will be called with the window as a single argument. If any of the CLOSEFNs are the atom DON'T, or if the value returned by any of the CLOSEFNs is the atom DON'T, the window will not be closed.

Note: A CLOSEFN should not call CLOSEW on its argument.

OPENFN

[Window Property]

The OPENFN window property can be a single function or a list of functions. If one of the OPENFNs is the atom DON'T, the window will not be opened. Otherwise, the OPENFNs are called after a window has been opened by OPENW (page 19.26), with the window as a single argument.

TOTOPFN

[Window Property]

If non-NIL, whenever the window is brought to the top, the TOTOPFN is called (with the window as a single argument). This function may be used to bring a collection of windows to the top together.

If the NOCALL TOPWFN argument of TOTOPW (page 19.26) is non-NIL, the TOTOPFN of the window is not called, which provides a way of avoiding in nite loops when using TOTOPW from within a TOTOPFN.

SHRINKFN

[Window Property]

The SHRINKFN window property can be a single function or a list of functions that are called just before a window is shrunk by SHRINKW (page 19.27), with the window as a single argument. If any of the SHRINKFNs are the atom DON'T,

## INTERLISP-D DISPLAY FACILITIES

or if the value returned by any of the CLOSEFNs is the atom DON'T, the window will not be shrunk.

ICONFN

[Window Property]

If SHRINKW (page 19.27) is called without being given a TO WHAT argument (as is the case when invoked from the SHRINK window command) and the window's ICONFN property is non-NIL, then it gets called with two arguments, the window being shrunk and the previously created icon, if any. The ICONFN should return one of the TO WHAT entities described on page 19.27 or return the previously created icon if it does not want to change it.

ICON

[Window Property]

If SHRINKW (page 19.27) is called without being given a TO WHAT argument, the window's ICONFN property is NIL, and the ICON property is non-NIL, then it is used as the value of TO WHAT.

ICONWINDOW

[Window Property]

Whenever an icon is created, it is cached on the property ICONWINDOW of the window, so calling SHRINKW again will reuse the same icon (unless overridden by the ICONFN).

Thus, to change the icon it is necessary to remove the ICONWINDOW property or call SHRINKW (page 19.27) explicitly giving a TO WHAT argument.

EXPANDFN

[Window Property]

The EXPANDFN window property can be a single function or a list of functions. If one of the EXPANDFNs is the atom DON'T, the window will not be expanded. Otherwise, the EXPANDFNs are called after the window has been expanded by EXPANDW (page 19.28), with the window as a single argument.

SCROLLFN

[Window Property]

If the SCROLLFN property is NIL, the window will not scroll. Otherwise, it should be a function of four arguments: (1) the window being scrolled, (2) the distance to scroll in the horizontal direction (positive to right, negative to left), (3) the distance to scroll in the vertical direction (positive up, negative down), and (4) a flag which is T if the scrolling button is being held down. For more information, see SCROLL.HANDLER (page 19.24). For most scrolling windows, the SCROLLFN function should be SCROLLBYREPAINTFN (page 19.24).

NEWREGIONFN

[Window Property]

The NEWREGIONFN is passed as the NEWREGIONFN argument to GETREGION (page 19.37) when the window is reshaped.

RESHAPEFN

[Window Property]

The RESHAPEFN window property can be a single function or a list of functions that are called when a window is reshaped by SHAPEW (page 19.26). If the RESHAPEFN is DON'T or a list containing DON'T, the window will not be reshaped. Otherwise, the function(s) are called after the window has been reshaped, its coordinate system readjusted to the new position, the title and border displayed, and the interior filled with texture. The RESHAPEFN should display any additional information needed to complete the window's image in the new position and shape. The RESHAPEFN is called with three arguments: (1) the window in its reshaped form, (2) a bitmap

## Miscellaneous Properties

with the contents of the old window, and (3) the region within the bitmap that contains the old image. This function is provided so that users can reformat window contents or whatever. `RESHAPEBYREPAINTFN` (page 19.33) is the default and should be useful for many windows.

`REPAINTFN`

[Window Property]

The `REPAINTFN` window property can be a single function or a list of functions that are called to repaint parts of the window by `REDISPLAYW` (page 19.27). The `REPAINTFN`s are called with two arguments: the window and the region in the coordinates of the window's display stream of the area that should be repainted. Before the `REPAINTFN` is called, the clipping region of the window is set to clip all display operations to the area of interest so that the `REPAINTFN` can display the entire window contents and the results will be appropriately clipped. (Note: `CLEARW` (page 19.27) should not be used in `REPAINTFN`s because it resets the window's coordinate system. If a `REPAINTFN` wants to clear its region `rst`, it should use `DSPFILL` (page 19.12).)

`MOVEFN`

[Window Property]

If the `MOVEFN` is `DON'T`, the window will not be moved by `MOVEW` (page 19.26). Otherwise, if the `MOVEFN` is non-`NIL`, it should be a function or a list of functions that will be called before a window is moved with two arguments: the window being moved and the new position of the lower left corner in screen coordinates. If the `MOVEFN` returns `DON'T`, the window will not be moved. If the `MOVEFN` returns a `POSITION`, the window will be moved to that position. Otherwise, the window will be moved to the specified new position.

`AFTERMOVEFN`

[Window Property]

If non-`NIL`, it should be a function or a list of functions that will be called after the window is moved (by `MOVEW`, page 19.26) with the window as an argument.

### 19.12.7.3 Miscellaneous Properties

`TITLE`

[Window Property]

Accesses the title of the window. If a title is added to a window whose title is `NIL` or the title is removed (set to `NIL`) from a window with a title, the window's exterior (its region on the screen) is enlarged or reduced to accommodate the change without changing the window's interior. For example, `(WINDOWPROP WINDO W 'TITLE "Results")` changes the title of `WINDO W` to be "Results". `(WINDOWPROP WINDO W 'TITLE NIL)` removes the title of `WINDO W`.

`BORDER`

[Window Property]

Accesses the width of the border of the window. The border will have at most 2 point of white (but never more than half) and the rest black. The default border is the value of the global variable `WBorder` (initially 4).

`EXTENT`

[Window Property]

Used to limit scrolling operations (see page 19.23). Accesses the extent region of the window. If non-`NIL`, the `EXTENT` is a region in the window's display stream that contains the complete image of the object being viewed by the window. User programs are responsible for updating the `EXTENT`. The functions `UNIONREGIONS`,



## INTERLISP-D DISPLAY FACILITIES

EXTENDREGION, etc. (page 19.3) are useful for computing a new extent region.

In some situations, it is useful to define an EXTENT that only exists in one dimension. This may be done by specifying an EXTENT region with a width or height of -1. SCROLLFN handling recognizes this situation as meaning that the negative EXTENT dimension is unknown.

PROCESS

[Window Property]

If the PROCESS window property is non-NIL, it should be a PROCESS and will be made the TTY process by GIVE.TTY.PROCESS (page 18.34), the default WINDOWENTRYFN property. This implements the mechanism by which the keyboard is associated with different processes.

PAGEFULLFN

[Window Property]

If the PAGEFULLFN is non-NIL, it will be called with the window as a single argument when the window is full (i.e., when enough has been printed since the last TTY interaction so that the next character printed will cause information to be scrolled off the top of the window.) If the PAGEFULLFN is NIL, the system function PAGEFULLFN (page 19.33) is called.

Note: PAGEFULLFN is only called on windows which are the TTYDISPLAYSTREAM of some process (see page 19.15).

The following properties are read-only (i.e. their property values cannot be changed using WINDOWPROP).

DSP

[Window Property]

Value is the display stream of the window. All system functions will operate on either the window or its display stream.

HEIGHT

[Window Property]

WIDTH

[Window Property]

Value is the height and width of the interior of the window (the usable space not counting the border and title).

REGION

[Window Property]

Value is a region (in screen coordinates) indicating where the window (counting the border and title) is located on the screen.

### 19.12.8 Auxiliary Functions

(RESHAPEBYREPAINTFN WINDOW OLDIMAGE OLDREGION)

[Function]

It BITBLTs the old region contents into the lower left corner of the new region. If the new shape is larger in either or both dimensions, the new areas exposed are to the top and right of the old image. When this happens, RESHAPEBYREPAINTFN calls WINDOW's REPAINTFN (page 19.32) to display the newly exposed region's contents. Note that this may result in two calls to the REPAINTFN.

(PAGEFULLFN WINDOW)

[Function]

If the window property PAGEFULLFN (page 19.33) is NIL, when the window is full the system function PAGEFULLFN is called. PAGEFULLFN simply returns if there are characters in the type-in buffer for WINDOW, otherwise it inverts the window

### Example: A Scrollable Window

and waits for the user to type a character. PAGEFULLFN is user advisable.

#### 19.12.9 Example: A Scrollable Window

The following is a simple example showing how one might create a scrollable window.

CREATE.PPWINDOW creates a window that displays the pretty printed expression EXPR. The window properties PPEXPR, PPORIGX, and PPORIGY are used for saving this expression, and the initial window position. Using this information, REPAINT.PPWINDOW simply reinitializes the window position, and prettyprints the expression again. Note that the whole expression is reformatted every time, even if only a small part actually lies within the window. If this window was going to be used to display very large structures, it would be desirable to implement a more sophisticated REPAINTFN that only redisplay that part of the expression within the window. However, this scheme would be satisfactory if most of the items to be displayed are small.

RESHAPE.PPWINDOW resets the window (and stores the initial window position), calls REPAINT.PPWINDOW to display the window's expression, and then sets the EXTENT property of the window so that SCROLLBYREPAINTFN will be able to handle scrolling and "thumbing" correctly.

```
(DEFINEQ

(CREATE.PPWINDOW
  [LAMBDA (EXPR)                                (* rrb "4-OCT- 82 12:06'")
                                              (* creates a window that displays
                                              a pretty printed expression.)

    (PROG (WINDOW)                             (* ask the user for a piece of the
                                              screen and make it into a window.)

      (SETQ WINDOW (CREATEW NIL "PP window"))
                                              (* put the expression on the
                                              property list of the window so that
                                              the repaint and reshape functions
                                              can access it.)

      (WINDOWPROP WINDOW (QUOTE PPEXPR)
                    EXPR)                    (* set the repaint and reshape
                                              functions.)

      (WINDOWPROP WINDOW (QUOTE REPAINTFN)
                    (FUNCTION REPAINT.PPWINDOW))

      (WINDOWPROP WINDOW (QUOTE RESHAPEFN)
                    (FUNCTION RESHAPE.PPWINDOW))
                                              (* make the scroll function
                                              SCROLLBYREPAINTFN, a system
                                              function that uses the repaint
                                              function to do scrolling.)

      (WINDOWPROP WINDOW (QUOTE SCROLLFN)
                    (FUNCTION SCROLLBYREPAINTFN))
                                              (* call the reshape function to
                                              initially print the expression and
                                              calculate its extent.)

      (RESHAPE.PPWINDOW WINDOW)
```

## INTERLISP-D DISPLAY FACILITIES

```
(RETURN WINDOW))
```

```
(REPAINT.PPWINDOW
 [LAMBDA (WINDOW REGION)          (* rrb "4-OCT- 82 11:52"*)

  (* the repainting function for a window with a pretty printed expression.
  This repainting function ignores the region to be repainted and repaints
  the entire window.)
```

```
                                (* set the window position to the
                                beginning of the pretty printing
                                of the expression.)
  (MOVETO (WINDOWPROP WINDOW (QUOTE PPORIGX))
    (WINDOWPROP WINDOW (QUOTE PPORIGY))
    WINDOW)
  (PRINTDEF (WINDOWPROP WINDOW (QUOTE PPEXPR))
    0 NIL NIL NIL WINDOW])
```

```
(RESHAPE.PPWINDOW
 [LAMBDA (WINDOW)                (* rrb "4-OCT- 82 12:01"*)
                                (* the reshape function for a
                                window with a pretty printed
                                expression.)
```

```
(PROG (BTM)

  (* set the position of the window so that the rst character appears in
  the upper left corner and save the X and Y for the repaint function.)
```

```
(DSPRESET WINDOW)
(WINDOWPROP WINDOW (QUOTE PPORIGX)
  (DSPXPOSITION NIL WINDOW))
(WINDOWPROP WINDOW (QUOTE PPORIGY)
  (DSPYPOSITION NIL WINDOW))
                                (* call the repaint function to
                                pretty print the expression in
                                the newly cleared window.)
(REPAINT.PPWINDOW WINDOW)
```

```
(* save the region actually covered by the pretty printed expression so
that the scrolling routines will know where to stop. The pretty printing
of the expression does a carriage return after the last piece of the
expression printed so that the current position is the base line of
the next line of text. Hence the last visible piece of the expression
(BTM) is the ending position plus the height of the font above the
base line e.g its ASCENT.)
```

```
(WINDOWPROP WINDOW (QUOTE EXTENT)
  (create REGION
    LEFT _ 0
```

## Interactive Display Functions

```

BOTTOM _[SETQ BTM (IPLUS (DSPYPOSITION NIL WINDOW)
                          (FONTPROP WINDOW (QUOTE ASCENT)

WIDTH _ (WINDOWPROP WINDOW (QUOTE WIDTH))
HEIGHT _ (IDIFFERENCE (WINDOWPROP WINDOW (QUOTE HEIGHT))

BTM)])
)

```

### 19.13 INTERACTIVE DISPLAY FUNCTIONS

The following functions allow the user to interactively specify positions or regions on the display screen.

(GETPOSITION WINDO W CURSOR ) [Function]  
 Returns a POSITION that is specied by the user. GETPOSITION waits for the user to press and release the left button of the mouse and returns the cursor position at the time of release. If WINDO W is a WINDOW, the position will be in the coordinate system of WINDO W 's display stream. If WINDO W is NIL, the position will be in screen coordinates. If CURSOR is a CURSOR, the cursor will be changed to it while GETPOSITION is running. If CURSOR is NIL, the value of the system variable CROSSHAIRS will be used as the cursor.

(GETBOXPOSITION WIDTH HEIGHT OR GX OR GY WINDO W PR OMPTMSG ) [Function]  
 Allows the user to position a "ghost" region of size WIDTH by HEIGHT on the screen, and returns the POSITION of the lower left corner of the region. If PR OMPTMSG is non-NIL, GETBOXPOSITION rst prints it in the PROMPTWINDOW. GETBOXPOSITION then changes the cursor to a box (using the global variable BOXCURSOR). If OR GX and OR GY are numbers, they are taken to be the original position of the region, and the cursor is moved to the nearest corner of that region. The user is then free to move the cursor around the screen. When a mouse button is depressed, a ghost region is locked to the cursor so that if the cursor is moved, the ghost region moves with it. If OR GX and OR GY are numbers, the corner of the original region that is nearest the cursor position at the time the button is pressed is locked, otherwise the lower left corner is locked. The user can change to another corner by continuing to hold down the left button and holding down the right button also. With both buttons down, the cursor can be moved across the screen without eect on the ghost region frame. When the right button is released, the mouse will snap to the nearest corner, which will then become locked to the cursor. When all buttons are released, the lower left corner of the region is returned. If WINDO W is a WINDOW, the returned position will be in WINDO W 's coordinate system; otherwise it will be in screen coordinates.

Example:

```

(GETBOXPOSITION 100 200 NIL NIL NIL
 "Specify the position of the command area.")

```

## INTERLISP-D DISPLAY FACILITIES

prompts the user for a 100 wide by 200 high region and returns its lower left corner in screen coordinates.

(GETREGION MINWIDTH MINHEIGHT INITREGION NEWREGIONFN NEWREGIONFNAR G) [Function]

Lets the user specify a new region and returns that region in screen coordinates. GETREGION prompts for a region by displaying a four-pronged box next to the cursor arrow. If the user presses the left button, one corner of a “ghost” region outline is locked to that point and the opposite corner is locked to the cursor. As the cursor moves, the outline expands. To specify a region, the user moves the cursor to one corner of the intended region, presses the left button, moves the cursor to the opposite corner while holding down the left button, and then releases the button.

If INITREGION is a REGION and the user presses the middle button, the corner of INITREGION farthest from the cursor position is xed and the corner nearest the cursor is locked to the cursor.

One can switch from one corner to another while positioning the region. To change to another corner, continue to hold down the left button and hold down the right button also. With both buttons down, the cursor can be moved across the screen without effect on the ghost region frame. When the right button is released, the cursor will snap to the nearest corner, which will become the moving corner. In this way, the region may be moved all over the screen, before its size and position is finalized.

MINWIDTH and MINHEIGHT, if given, are the smallest WIDTH and HEIGHT that the returned region will have. If the user specified region is smaller, it will be increased in width or height to these dimensions.

If NEWREGIONFN is non-NIL, it will be called to determine values for the positions of the corners. This provides a way of “filtering” prospective regions; for instance, by restricting the region to lie on an arbitrary grid. When the user is specifying a region, the region is determined by two of its corners, one that is xed and one that is tracking the cursor. Each time the cursor moves or a mouse button is pressed, NEWREGIONFN is called with three arguments: FIXEDPOINT, the position of the xed corner of the prospective region; MOVINGPOINT, the position of the opposite corner of the prospective region; and NEWREGIONFNARG. NEWREGIONFNARG allows the caller of GETREGION to pass information to the NEWREGIONFN. The first time a button is pressed, MOVINGPOINT is NIL and FIXEDPOINT is the position the user selected for the xed corner of the new region. In this case, the position returned by NEWREGIONFN will be used for the xed corner instead of the one proposed by the user. For all other calls, FIXEDPOINT is the position of the xed corner (as returned by the previous call) and MOVINGPOINT is the new position the user selected for the opposite corner. In these cases, the value of NEWREGIONFN is used for the opposite corner instead of the one proposed by the user. In all cases, the ghost region is drawn with the values returned by NEWREGIONFN.

(GETBOXREGION WIDTH HEIGHT OR GX OR GY WINDOW PROMPTMSG) [Function]

Performs the same prompting as GETBOXPOSITION and returns the REGION specified by the user instead of the POSITION of its lower left corner.

## Menus

### 19.14 MENUS

A menu is basically a means of selecting from a list of items. The system provides common layout and interactive user selection mechanisms, then calls a user-supplied function when a selection has been confirmed. The two major constituents of a menu are a list of items and a “when selected function.” The label that appears for each item is the item itself for non-lists, or its CAR if the item is a list. The menu includes a position on the screen where it will be displayed and a means of specifying the place in the menu that is to be put at that position. In addition, there are a multitude of different formatting parameters for specifying font, size, and layout. When a menu is created, its unspecified fields are filled with defaults and its screen image is computed and saved.

Menus can be either pop up or fixed. If fixed menus are used, the menu must be included in a window.

(MENU MENU POSITION ) [Function]  
This function provides menus that pop up when they are used. It displays MENU at POSITION (in screen coordinates) and waits for the user to select an item with a mouse key. While any key is down, the selected menu item is video reversed. When all keys are released, MENU's WHENSELECTEDFN field is called with three arguments: (1) the item selected, (2) the menu, and (3) the last mouse key released (LEFT, MIDDLE, or RIGHT), and MENU returns its value. If no item is selected, MENU returns NIL. If POSITION is NIL, the menu is brought up at the value from MENU's MENUPOSITION field, if it is a POSITION, or at the current cursor position. The orientation of MENU with respect to the specified position is determined by its MENUOFFSET field.

(ADDMENU MENU WINDOW POSITION \_ ) [Function]  
This function provides menus that remain active in windows. ADDMENU displays MENU at POSITION in WINDOW (POSITION is defaulted as in MENU except that it is in window coordinates). MENU is added to the MENU property of WINDOW. The CURSORINFN and BUTTONEVENTFN of WINDOW are replaced with MENUBUTTONFN, so that MENU will be active during TTY wait. RESHAPEFN of WINDOW is set to restore MENU's image when the window is reshaped. When an item is selected, the value of the WHENSELECTEDFN field of MENU is called with three arguments: (1) the item selected, (2) the menu, and (3) the mouse key that the item was selected with (LEFT, MIDDLE, or RIGHT). More than one menu can be put in a window, but a menu can only be added to one window at a time. If WINDOW is not given, a window is created at POSITION (in screen coordinates) that is the size of MENU.

ADDMENU returns the window into which MENU is placed.

(DELETEMENU MENU CLOSEFLAG FROMWINDOW W ) [Function]  
This function removes MENU from the window FROMWINDOW W. If MENU is the only menu in the window and CLOSEFLAG is non-NIL, its window will be closed (by CLOSEW).

If FROMWINDOW W is NIL, the list of currently active (open) windows is searched for one that contains MENU. If none is found, DELETEMENU does nothing.

## INTERLISP-D DISPLAY FACILITIES

### 19.14.1 Menu Fields

A menu is a datatype with the following fields:

ITEMS	[Menu Field] The list of items to appear in the menu. If an item is a list, its CAR will appear in the menu. If the item (or its CAR) is a bitmap, the bitmap will be displayed in the menu. The default selection functions interpret each item as a list of three elements: a label, a form whose value is returned upon selection, and a help string that is printed in the prompt window when the user presses a mouse key with the cursor pointing to this item.
WHENSELECTEDFN	[Menu Field] A function to be called when an item is selected. The function is called with three arguments: (1) the item selected, (2) the menu, and (3) the mouse key that the item was selected with (LEFT, MIDDLE, or RIGHT). The default function DEFAULTWHENSELECTEDFN evaluates and returns the value of the CADR of the item if there is one, or simply returns the item if it is not a list or if its CADR is NIL.
WHENHELDNFN	[Menu Field] The function which is called when the user has held a mouse key on an item for MENUHELDWAIT milliseconds (initially 1200). The function is called with three arguments: (1) the item selected, (2) the menu, and (3) the mouse key that the item was selected with (LEFT, MIDDLE, or RIGHT). WHENHELDNFN is intended for prompting users. The default is DEFAULTMENUHELDNFN which prints (in the prompt window) the third element of the item or, if there is not a third element, the string "This item will be selected when the button is released."
WHENUNHELDNFN	[Menu Field] If WHENHELDNFN was called, WHENUNHELDNFN will be called: (1) when the cursor leaves the item, (2) when a mouse key is released, or (3) when another key is pressed. The function is called with the same three argument values used to call WHENHELDNFN. The default WHENUNHELDNFN is the function CLR_PROMPT (page 19.19), which just clears the prompt window.
MENUPOSITION	[Menu Field] The position of the menu to be used if the call to MENU or ADDMENU does not specify a position. For popup menus, this is in screen coordinates. For xed menus, it is in the coordinates of the window the menu is in. The point within the menu image that is placed at this position is determined by MENUOFFSET. If MENUPOSITION is NIL, the menu will be brought up at the cursor position.
MENUOFFSET	[Menu Field] The position in the menu image that is to be located at MENUPOSITION. The default offset is (0,0). For example, to bring up a menu with the cursor over a particular menu item, set its MENUOFFSET to a position within that item and set its MENUPOSITION to NIL.
MENUFONT	[Menu Field] The font in which the items will be appear in the menu. Default is the value of

## Menu Fields

MENUFONT, initially Helvetica 10.

**TITLE** [Menu Field]  
If specified, a title will appear in a line above the menu. The title will be in the same font as window titles.

**CENTERFLG** [Menu Field]  
If non-NIL, the menu items are centered; otherwise they are left-justified.

**MENUROWS** [Menu Field]  
**MENUCOLUMNS** [Menu Field]  
These fields control the shape of the menu in terms of rows and columns. If MENUROWS is given, the menu will have that number of rows. If MENUCOLUMNS is given, the menu will have that number of columns. If only one is given, the other one will be calculated to generate the minimal rectangular menu. (Normally only one of MENUROWS or MENUCOLUMNS is given.) If neither is given, the items will be in one column.

**ITEMHEIGHT** [Menu Field]  
The height of each item box in the menu. If not specified, it will be the maximum of the height of the MENUFONT and the heights of any bitmaps appearing as labels.

**ITEMWIDTH** [Menu Field]  
The width of each item box in the menu. If not specified, it will be the width of the largest item in the menu.

**MENUBORDERSIZE** [Menu Field]  
The size of the border around each item box. If not specified, 0 (no border) is used.

**MENUOUTLINESIZE** [Menu Field]  
The size of the outline around the entire menu. If not specified, a maximum of 1 and the MENUBORDERSIZE is used.

**CHANGEOFFSETFLG** [Menu Field]  
(popup menus only) If CHANGEOFFSETFLG is non-NIL, the position of the menu offset is set each time a selection is confirmed so that the menu will come up next time in the same position relative to the cursor. This will cause the menu to reappear in the same place on the screen if the cursor has not moved since the last selection. This is implemented by changing the MENUOFFSET field on each use. If CHANGEOFFSETFLG is the atom X or the atom Y, only the X or the Y coordinate of the MENUOFFSET field will be changed. For example, by setting the MENUOFFSET position to (-1,0) and setting CHANGEOFFSETFLG to Y, the menu will pop up so that the cursor is just to the left of the last item selected. This is the setting of the window command menus.

The following fields are read only.

**IMAGEHEIGHT** [Menu Field]  
Returns the height of the entire menu.



## INTERLISP-D DISPLAY FACILITIES

IMAGEWIDTH [Menu Field]  
Returns the width of the entire menu.

### 19.14.2 Miscellaneous Menu Functions

(WFROMMENU MENU ) [Function]  
Returns the window MENU is located in, if it is in one; NIL otherwise.

(DOSELECTEDITEM MENU ITEM BUTTON ) [Function]  
Calls MENU 's WHENSELECTEDFN on ITEM and BUTTON . It provides a programmatic way of making a selection. It does not change the display.

(MENUITEMREGION ITEM MENU ) [Function]  
Returns the region occupied by ITEM in MENU .

(SHADEITEM ITEM MENU SHADE DSOR W ) [Function]  
Shades the region occupied by ITEM in MENU . If DSOR W is a display stream or a window, it is assumed to be where MENU is displayed. Otherwise, WFROMMENU is called to locate the window MENU is in.

### 19.14.3 Examples of Menu Use

```
(create MENU ITEMS _ '((YES T) (NO)) )
```

Creates a menu with items YES and NO in a single vertical column. If YES is selected, T will be returned. Otherwise, NIL will be returned.

```
(create MENU ITEMS _ '(1 2 3 4 5 6 7 8 9 * 0 #)
      CENTERFLG _ T
      MENUCOLUMNS _ 3
      MENUFONT _ (FONTCREATE 'HELVETICA 10 'BOLD)
      ITEMHEIGHT _ 15
      ITEMWIDTH _ 15
      CHANGEOFFSETFLG _ T)
```

Creates a touch-tone-phone number pad with the items in 15 by 15 boxes printed in Helvetica 10 bold font. If used in pop up mode, its rst use will have the cursor in the middle. Subsequent use will have the cursor in the same relative location as the previous selection.

```
(SELECTQ [MENU
  (COND ((type? MENU FOOMENU)
    (* use previously computed menu.)
    FOOMENU)
    (T (* create and save the menu)
      (SETQ FOOMENU
        (create MENU
          ITEMS _ '((A 'A-SELECTED "prompt string for A")
                    (B 'B-SELECTED "prompt string for B")
                    (A-SELECTED (* if A is selected) (DOATHING))
```

## Grid Functions

```
(B-SELECTED (* if B is selected) (DOETHING))
(PROGN (* user selected outside the menu) NIL))
```

This expression displays a pop up menu with two items, A and B, and waits for the user to select one. If A is selected, DOETHING is called. If B is selected, DOETHING is called. If neither of these is selected, the form returns NIL.

The purpose of this example is to show some good practices to follow when using menus. First, the menu is only created once, and saved in the variable FOOMENU. This is more efficient if the menu is used more than once. Second, all of the information about the menu is kept in one place, which makes it easy to understand and edit. Third, the forms evaluated as a result of selecting something from the menu are part of the code and hence will be known to masterscope (as opposed to the situation if the forms were stored as part of the items). Fourth, the items in the menu have help strings for the user. Finally, the code is commented (always worth the trouble).

### 19.15 GRID FUNCTIONS

A Grid is a partitioning of an arbitrary coordinate system (hereafter referred to as the “source system”) into rectangles. This subsection 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 GridSpec) which is the origin rectangle of the Grid in terms of the source system. Its LEFT is the X-coordinate of the left edge of the origin rectangle, its BOTTOM 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 UNITSWIDE UNITSHIGH GRIDBORDER DISPLAYSTREAM GRIDSHADE ) [Function]
  Outlines the grid defined by GRIDSPEC which is UNITSWIDE rectangles wide and
  UNITSHIGH rectangles high on DISPLAYSTREAM . Each box in the grid has a border
  within it that is GRIDBORDER points on each side; so the resulting lines in the grid
  are 2*GRIDBORDER thick. If GRIDBORDER 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 in that shade.
```

```
(SHADEGRIDBOX X Y SHADE OPERATION GRIDSPEC GRIDBORDER DISPLAYSTREAM ) [Function]
  Shades the grid rectangle (X,Y) of GRIDSPEC with texture SHADE using OPERATION
  on DISPLAYSTREAM . 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 .
```

## INTERLISP-D DISPLAY FACILITIES

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

(LEFTOFGRIDCOORD GRID X GRIDSPEC ) [Function]  
Returns the source system X-coordinate of the left edge of a Grid rectangle at Grid X-coordinate GRID X (in the Grid specified by GRIDSPEC ).

(BOTTOMOFGRIDCOORD GRID Y GRIDSPEC ) [Function]  
Returns the source system Y-coordinate of the bottom edge of a Grid rectangle at Grid Y-coordinate GRID Y (in the Grid specified by GRIDSPEC ).

### 19.16 COLOR GRAPHICS

*Note: This section describes the Interlisp-D facilities for using a color display. To use these facilities you need to have a Xerox 1100 or Xerox 1132 with a color display attached, and you must load in the LispUsers les COLOR.DCOM and LLCOLOR.DCOM (automatically loaded by COLOR.DCOM).*

The color boards on the Xerox 1100 and the Xerox 1132 differ in design. The Xerox 1100 board supports 4 bits per pixel color. The Xerox 1132 supports 4 or 8 bits per pixel. All of the user's code should be written in higher level machine independent functions.

Both color boards produce an image that is 640 pixels wide by 480 pixels high. The image can be thought of as a paint-by-number painting where the number of a pixel is its value. The number of bits per pixel (4 on the Xerox 1100, 4 or 8 on the Xerox 1132) determines the number of different colors that can be displayed at one time. When there are 4 bpp, 16 colors can be displayed at once. When there are 8 bpp, 256 colors can be displayed at once. A mapping table called a "color map" determines what color actually appears for each pixel value. A color map gives the color in terms of how much of the three primary colors (red, green and blue) displayed on the screen for each possible pixel value. In the following sections, the notions of "color map", and "color" are described.

#### 19.16.1 Color Bitmaps

A "color bitmap" is actually just a bitmap that allows more than one bit per pixel. To test whether a bitmap x is a "color bitmap", use the following form:

(NEQ (fetch (BITMAP BITMAPBITSPERPIXEL) of x) 1)

Color bitmaps are created by calling BITMAPCREATE (page 19.4) with a BITSPERPIXEL argument of anything other than 1 or NIL. Currently, any value of BITSPERPIXEL except 1, 4, 8 or NIL (defaults to 1) will cause an error.

A 4 bit per pixel color screen bitmap uses approximately 76k of storage. There is only one such bitmap. The following function provides access to it:

(COLORSCREENBITMAP) [Function]  
Returns the color bitmap that is being or will be displayed on the color display. This will be NIL if the color display has never been turned on (see COLORDISPLAY, page 19.47).

## Color Specifications

WHOLECOLORDISPLAY	[Variable]
A global variable set to a REGION that covers the entire color display screen. Currently this is (CREATEREGION 0 0 640 480).	
COLORSCREENWIDTH	[Variable]
The width of the color display. Currently 640.	
COLORSCREENHEIGHT	[Variable]
The height of the color display. Currently 480.	

### 19.16.2 Color Specifications

A color map maps a color number (from 0 to  $2^{\text{BITS PER PIXEL}} - 1$ ) into the intensities of the three color guns (red, green and blue). Each entry in the color map has 8 bits for each of the primary colors allowing 256 levels per primary or  $2^{24}$  possible colors (not all of which are distinct to the human eye). Within Interlisp-D programs, colors can be manipulated as numbers, red-green-blue triples, names, or hue-lightness-saturation triples. Any function that takes a color will accept any of the different specifications.

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

A red-green-blue (RGB) triple is a list of three numbers between 0 and 255. The first element gives the intensity for RED, the second for GREEN and the third for BLUE. When an RGB triple is used, the current color map is searched to find the color with the correct intensities. If none is found, an error is generated. (That is, no attempt is made by the system to assign color numbers to intensities automatically.) Example of an RGB triple is (255 255 255) which gives the color white. The record RGB with fields RED, GREEN, and BLUE is provided to manipulate RGB triples.

A color name is an atom that is on the association-list COLORNAMES. The CDR of the color name's entry will be used as the color corresponding to the color name. This can be any of the other representations. (Note: It can even be another color name. Loops in the name space such as would be caused by putting '(RED . CRIMSON) and '(CRIMSON . RED) on COLORNAMES are *not* checked for by the system.) Several color names are available in the initial system and are intended to allow color programs written by different users to coexist. These are:

## INTERLISP-D DISPLAY FACILITIES

name	RGB	number in default color map
BLACK	(0 0 0)	0
BLUE	(0 0 255)	1
GREEN	(0 255 0)	2
CYAN	(0 255 255)	3
RED	(255 0 0)	4
MAGENTA	(255 0 255)	5
YELLOW	(255 255 0)	6
WHITE	(255 255 255)	7

A hue-lightness-saturation triple is a list of three numbers. The first number (hue) is between 0 and 355 and indicates a position in degrees on a color wheel (blue at 0, red at 120 and green at 240). The second (lightness) is a FLOATP between 0 and 1 which indicates how much total intensity is in the color. The third (saturation) is a FLOATP between 0 and 1 which indicates how disparate the three primary levels are. The record HLS with fields HUE, LIGHTNESS, and SATURATION is provided to manipulate HLS triples. Example: the color blue is represented in HLS notation by (0 .5 1.0).

(COLORNUMBERP COLOR BITSPERPIXEL NOERRFLAG) [Function]  
 Returns the color number (set into the screen color map) of COLOR. COLOR should be either (1) a positive number less than the maximum number of colors, (2) a color name, (3) an RGB triple, or (4) an HLS triple. If COLOR is one of the above and is found in the screen colormap, its color number in the screen color map is returned. If not, an error is generated unless NOERRFLAG is non-NIL, in which case NIL is returned.

(RGBP X) [Function]  
 Returns X if X is an RGB triple; NIL otherwise.

(HLSP X) [Function]  
 Returns X if X is an HLS triple; NIL otherwise.

### 19.16.3 Color Maps

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

(COLORMAPCREATE INTENSITIES BITSPERPIXEL) [Function]  
 Creates a color map for a screen that has BITSPERPIXEL bits per pixel. If BITSPERPIXEL is NIL, the number of bits per pixel is taken from the current color display setting. INTENSITIES specifies the initial colors that should be in the map. If INTENSITIES is not NIL, it should be a list of color specifications

## Color Maps

(other than color numbers), e.g. the list of RGB triples returned by the function `INTENSITIESFROMCOLORMAP` (below). If `INTENSITIES` is `NIL`, the default is the value of `\DEFAULTCOLORINTENSITIES` (if `BITSPERPIXEL` is 4) or `\DEFAULT8BITCOLORINTENSITIES` (if `BITSPERPIXEL` is 8).

`(COLORMAPP COL ORMAP? BITSPERPIXEL )` [Function]  
Returns `COL ORMAP?` if it is a color map that has `BITSPERPIXEL` bits per pixel; `NIL` otherwise. If `BITSPERPIXEL` is `NIL`, it returns `COL ORMAP?` if it is either a 4 bits per pixel or an 8 bits per pixel colormap.

`(INTENSITIESFROMCOLORMAP COL ORMAP )` [Function]  
Returns a list of the intensity levels of `COL ORMAP` (default is `(SCREENCOLORMAP)`) in a form accepted by `COLORMAPCREATE`. This list can be written on `le` and thus provides a way of saving color map specifications.

`(COLORMAPCOPY COL ORMAP BITSPERPIXEL )` [Function]  
If `COL ORMAP` is a color map, it returns a color map that contains the same color intensities as `COL ORMAP`; otherwise it returns a color map with default color values.

`(SCREENCOLORMAP NEW COL ORMAP )` [Function]  
Reads and sets the color map that is used by the color display. If `NEW COL ORMAP` is non-`NIL`, it should be a color map and `SCREENCOLORMAP` sets the system color map to be that color map. Returns the previous value of the screen color map. If `NEW COL ORMAP` is `NIL`, the current screen color map is returned without change.

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

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

`(SETCOLORINTENSITY COL ORMAP COL ORNUMBER COL ORSPEC )` [Function]  
Sets the primary intensities of color number `COL ORNUMBER` in the color map `COL ORMAP` to the ones specified by `COL ORSPEC`. `COL ORSPEC` can be either an RGB triple, an HLS triple or a color name. Returns `NIL`.

`(COLORLEVEL COL ORMAP COL ORNUMBER PRIMAR YCOL OR NEWLEVEL )` [Function]  
Sets and reads the intensity level of the primary color `PRIMAR YCOL OR` (either `RED`, `GREEN` or `BLUE`) for the color number `COL ORNUMBER` in the color map `COL ORMAP`. If `NEWLEVEL` is a number between 0 and 255, it is set. The previous value of the intensity of `PRIMAR YCOL OR` is returned.

`(ADJUSTCOLORMAP PRIMAR YCOL OR DELTA COL ORMAP )` [Function]  
Adds `DELTA` to the intensity of the primary color `PRIMAR YCOL OR` (either `RED`, `GREEN` or `BLUE`) for every color number in `COL ORMAP`.

`(ROTATECOLORMAP COL ORMAP STARTCOL OR THRCOL OR )` [Function]  
Rotates a sequence of colors in `COL ORMAP`. The rotation moves the intensity values of color number `STARTCOL OR` into color number `STARTCOL OR +1`, the intensity values of color number `STARTCOL OR +1` into color number `STARTCOL OR +2`, etc. and `THRCOL OR`'s values into `STARTCOL OR`.

## INTERLISP-D DISPLAY FACILITIES

(EDITCOLORMAP VAR NOQFL G)

[Function]

Allows interactive editing of a color map. If VAR is an atom whose value is a color map, its value is edited. Otherwise a new color map is created and edited. The color map being edited is made the screen color map while the editing is taking place so that its effects can be observed. The edited color map is returned as the value.

If NOQFL G is NIL and the color display is on, the user is asked if they want a test pattern of colors. A yes response will cause the function SHOWCOLORTESTPATTERN to be called which will display a test pattern with blocks of each of the possible colors.

The user is prompted for the location of a color control window to be placed on the black and white display. This window allows the value of any of the colors to be changed. The color number of the color being edited is in the upper left part of the window. Six bars are displayed. The right three bars give the color intensities for the three primary colors of the current color number. The left three bars give the value of the color's Hue, Lightness and Saturation parameters. These levels can be changed by positioning the cursor in one of the bars and pressing the LEFT button. While the LEFT button is down, the value of that parameter will track the Y position of the cursor. When the LEFT button is released, the color tracking stops. The color being edited is changed by pressing the MIDDLE button while the cursor is in the interior of the edit window. This will bring up a menu of color numbers. Selecting one sets the current color to the selected color.

The color being edited can also be changed by selecting the menu item "PickPt". This will switch the cursor onto the color screen and allow the user to select a point from the color screen. It will then edit the color of the selected point.

To stop the editing, move the cursor into the title of the editing window and press the MIDDLE button. This will bring up a menu. Select STOP to quit.

### 19.16.4 Turning the Color Display On and Off

The color display can be turned on and off. While the color display is on, the memory used for the color display screen bitmap is locked down and a significant amount of processing time (35% on the Xerox 1100) is used to drive the color display.

(COLORDISPLAYP)

[Function]

Returns the current color map if the color display is on; otherwise NIL.

(COLORDISPLAY COLORMAP BITSPERPIXEL CLEARSCREENFL G)

[Function]

If COLORMAP is NIL, it turns off the color display. If COLORMAP is non-NIL, it turns on the color display allocating BITSPERPIXEL bits per pixel. If COLORMAP is a color map, it is used as the screen color map. If CLEARSCREENFL G is non-NIL, all of the bits in the color screen are set to 0.

Turning on the color display requires allocating and locking down the memory necessary to hold the color display screen bitmap and the system color map. Turning the color display off frees this memory.

## Printing and Drawing in Color

### 19.16.5 Printing and Drawing in Color

The current color implementation allows display streams to operate on color bitmaps. The following two functions set the color in which a display stream prints or draws:

(DSPCOLOR COL OR DISPLAYSTREAM ) [Function]  
Sets the foreground color of a display stream. Returns the previous foreground color. If COL OR is NIL, it returns the current foreground color without changing anything. The default foreground color is 7, which is white in the default color map.

(DSPBACKCOLOR COL OR DISPLAYSTREAM ) [Function]  
Sets the background color of a display stream. Returns the previous background color. If COL OR is NIL, it returns the current background color without changing anything. The default background color is 0 which is black in the default color map.

BITBLT, the line and curve drawing routines and the printing routines know how to operate on a display stream that has a color bitmap as its destination. Following are some notes about them.

BITBLT (page 19.4) When BITBLTing from a color bitmap onto another color bitmap with the same bits per pixel, the operations PAINT, INVERT and ERASE are done on a bit level; not on a pixel level. Thus painting color 3 onto color 10 will result in color 11.

When BITBLTing from a black and white bitmap onto a color bitmap, the 1 bits will appear in the DSPCOLOR and the 0 bits in DSPBACKCOLOR. Currently, REPLACE is the only operation that is supported BITBLTing from black and white to color. This operation is fairly expensive; if the same bitmap is going to be put up several times in the same color it is faster to create a color copy then blt the color copy.

If the SOURCE is TEXTURE and the DESTINATION is a color bitmap, the TEXTURE argument is taken to be a color. Thus, to fill an area with the color BLUE, do:

```
(BITBLT NIL NIL NIL COLORBITMAP 50 75 100 200 'TEXTURE 'REPLACE  
'BLUE)
```

Curve drawing (page 19.14)

For the functions DRAWCIRCLE, DRAWELLIPSE and DRAWCURVE, the notion of a brush has been extended to include a color. A brush can be a list of the form (SHAPE SIZE COL OR). A brush can also be a bitmap, which can be color bitmap.

Line drawing (page 19.13)

The line drawing functions have been extended to take another argument which is the color the line is to appear in if the destination of the display stream is a color bitmap. If the COL OR argument is NIL, the DSPCOLOR of the display stream is used.

Printing

Printing only works (currently) in REPLACE mode. The characters will have a foreground color of DSPCOLOR and a background of DSPBACKCOLOR. The first time a character is printed in a new color, the color images corresponding to the



## INTERLISP-D DISPLAY FACILITIES

current font are calculated and cached. Thus the first character may take a while to appear but succeeding characters print quickly.

### 19.16.6 Using the Cursor on the Color Screen

The cursor can be moved to the color screen. While on the color screen, the cursor is placed using XOR mode, thus with some color maps it may be hard to see. It is automatically taken down whenever an operation is performed that changes any bits on the color screen. While the cursor is on the color screen, the black and white cursor is cleared.

(CHANGECURSORSCREEN SCREENBITMAP ) [Function]  
SCREENBITMAP must be either the value of (COLORSCREENBITMAP) or the value of (SCREENBITMAP). CHANGECURSORSCREEN moves the cursor onto the specified screen. The value returned is the screen bitmap that the cursor was on before CHANGECURSORSCREEN was called.

### 19.16.7 Miscellaneous Color Functions

The following functions provide some common operations on color bitmaps and display streams.

(COLORFILL REGION COLORNUMBER COLORBITMAP OPERATION ) [Function]  
Fills the region REGION in COLORBITMAP with the color COLORNUMBER, using the operation OPERATION.

(COLORFILLAREA LEFT BOTTOM WIDTH HEIGHT COLORNUMBER COLORBITMAP OPERATION ) [Function]  
Fills an area in the color bitmap with a color.

(COLORIZEBITMAP BITMAP 0COLOR 1COLOR BITSPERPIXEL ) [Function]  
Creates and returns a color bitmap copying the black and white bitmap BITMAP. The returned color bitmap will have color number 1COLOR in those pixels of BITMAP that were 1 and 0COLOR in those pixels of BITMAP that were 0. This provides a way of producing a color bitmap from a black and white bitmap. Note: this is a fairly expensive operation in terms of both time and space.

### 19.16.8 Demonstration programs

*The following functions provide some demonstrations of the color display. These are available in the Lispuser's file COLORDEMO.DCOM.*

(COLORDEMO) [Function]  
Brings up a menu of color demonstration programs. The system will cycle through the entries on the menu automatically, allowing each to run for a small fixed amount of time (typically 40 seconds). Selecting one of the entries in the menu will cause it to start that program.

(COLORDEMO1) [Function]  
Runs the Interlisp-D logo demonstration until a button is pressed then adds

## Demonstration programs

COLORKINETIC. The MIDDLE button will bring up a menu that allows changing the speed of rotation or editing the color map. The LEFT button will rotate the color map in the kinetic area.

(COLORDEMO2 SIZE) [Function]  
Puts up a test pattern of size SIZE, then rotates the color map. The speed of rotation of the color map is determined by the Y position of the cursor. The MIDDLE button will bring up a menu that allows editing of the color map or changing the color map to a map of different shades of one color.

(COLORKINETIC REGION FIRSTCOL OR LASTCOL OR ) [Function]  
Runs color kinetic in a region REGION of the color display using colors FIRSTCOL OR through LASTCOL OR .

(TUNNEL SPEED ) [Function]  
Draws a series of concentric rectangles of increasing size in increasing color numbers. SPEED determines the size of the rectangles. This can then be “run” by calling ROTATEIT described below.

(MINESHAFT N OUTFL G ) [Function]  
Draws a series of concentric rectangles of size N in increasing color numbers. OUTFL G determines whether the color numbers increase or decrease. This can then be “run” by calling ROTATEIT described below.

(WELL N ) [Function]  
Draws a series of concentric circles on the color screen in increasing color numbers. The circles will be of size N. This can then be “run” by calling ROTATEIT described below.

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

(ROTATEIT BEGINCOL OR ENDCOL OR WAIT ) [Function]  
Goes into an infinite loop rotating the screen color map. The colors between BEGINCOL OR (default 0) and ENDCOL OR (default maximum color) are rotated. If WAIT is given, (DISMISS WAIT) is called each time the color map is changed. This provides an easy way of “animating” screen images.

*Note: The following function is available in the Lispusers file COLORPOLYGONS.DCOM.*

(COLORPOLYDEMO COL ORDS ) [Function]  
Runs a version of the Polygons program on the color screen.