



XEROX

# ***Artificial Intelligence Systems***

***Artificial Intelligence  
Systems***

*Lisp User Packages*

*Lisp User Packages*

# LISPUSERS' MODULES MANUAL



en·vos

309000  
Medley Release  
September 1988

---

Address comments to:  
Envos Corporation  
1157 San Antonio Road  
Mountain View, CA 94043

LISPUSERS' MODULES MANUAL

309000

Medley Release

September, 1988

Copyright © 1988 by Envos Corporation.

All rights reserved.

Envos is a trademark of Envos Corporation

Xerox® is a registered trademark of Xerox Corporation.

UNIX® is a registered trademark of AT&T Bell Laboratories.

Copyright protection includes material generated from the software programs displayed on the screen, such as icons, screen display looks, and the like.

---

The information in this document is subject to change without notice and should not be construed as a commitment by Envos Corporation. While every effort has been made to ensure the accuracy of this document, Envos Corporation assumes no responsibility for any errors that may appear.

TABLE OF CONTENTS

ACE .....	1
ADDRESSBOOK .....	8
AIREGIONS.....	11
ANALYZER .....	19
BACKGROUNDIMAGES .....	20
BACKGROUNDMENU .....	23
BITMAPFNS.....	26
CALENDAR.....	27
CANVASCONVERTER.....	34
CD.....	36
CHATEMACS.....	38
CHATSERVER .....	40
CHECKPOINT.....	43
CL-TTYEDIT .....	44
COMPARESOURCES.....	45
COMPARETEXT.....	48
COMPILEBANG .....	50
COURIERDEFS .....	51
COURIEREVALSERVE .....	52
COURIERIMAGESTREAM.....	53
COURIERSERVE.....	55
CROCK.....	58
DATEFORMAT-EDITOR.....	60
DEFAULTSUBITEMFN.....	62
DIGI-CLOCK.....	63
DOC-OBJECTS .....	65
DONZ .....	68
DSPSCALE .....	70
EDITBG.....	72
EDITKEYS .....	73
EQUATIONS .....	74
EQUATION EDITOR PROGRAMMERSGUIDE .....	77
EQUATIONEXAMPLES .....	85
ETHERBOOT.....	87
FILEWATCH.....	89
FILLREGION .....	92
FTPSEVER-MULTI-CONNECTIONS .....	94
GRAPHCALLS .....	95
GREP.....	100
GRID-ICONS .....	101

HANOI.....	104
HASHBUFFER.....	105
HASHDATUM.....	107
HEADLINE .....	108
HPGL .....	109
IDLEHAX .....	111
INSPECTCODE-TEDIT .....	113
KEYOBJ .....	115
KINETIC .....	116
KOTOLOGO .....	117
LIFE.....	118
LOADMENUITEMS.....	119
LOGIC.....	122
LOOKUPINFILES.....	130
MAGNIFIER .....	132
MAKEGRAPH .....	133
MANAGER .....	137
MATHTONS .....	143
MICROTEK .....	144
MONITOR.....	152
NEATICONS.....	154
NOTEPAD.....	158
NSCOPYFILE.....	161
NSDISPLAYSIZES.....	162
NSPROTECTION .....	164
PAGEHOLD .....	169
PIECE-MENUS .....	172
PLOT.....	174
PLOTEXAMPLES.....	188
PLOTOBJECTS.....	189
PLOTOBJECTS1.....	194
POSTSCRIPT .....	196
PS-SEND .....	200
PS-TTY.....	201
PREEMPTIVE .....	202
PRESSFROMNS.....	203
PRETTYFILEINDEX.....	206
PRINTERMENU.....	215
PROGRAMCHAT .....	216
PROMPTREMINDERS.....	217
PROOFREADER .....	220
QEDIT .....	223
READAIS .....	226

READAPPLEFONT.....	228
READBRUSH.....	229
READDATATYPE.....	230
READDISPLAYFONT.....	231
REGION.....	232
REMOTEPSW.....	234
RPC.....	235
SCREENPAPER.....	244
SEDIT-MENU-ALWAYS.....	245
SETDEFAULTPRINTER.....	246
SHOWTIME.....	247
SIMPLECHAT.....	249
SOLID-MOVEW.....	250
SOLITAIRE.....	252
STARBG.....	253
STEP-COMMAND-MENU.....	255
STORAGE.....	256
SYSTATS.....	258
TALK.....	259
TCPTIME.....	265
TEDITKEY.....	267
TILED-SEDIT.....	273
TRAJECTORY-FOLLOWER.....	275
TRICKLE.....	276
TURBO-WINDOWS.....	277
TWODGRAPHICS.....	281
UNBOXEDOPS.....	285
UUECODE.....	288
VSTATS.....	289
WDWHACKS.....	293
WHO-LINE.....	294
WHOCALLS.....	299
XCL-BRIDGE.....	300

[This page intentionally left blank]

---

---

**ACE**

---

---

By: Michel Denber (Denber.wbst@Xerox.com) Compiled for Medley by Larry Masinter  
(Masinter.PA@Xerox.COM)

**Files: ACE.LCOM**

**Data files: ACE-APPLEDEMO.ACE, ACE-BOUNCINGBALL.ACE, ACE-FOUETTE.ACE**

## **Animation Compiler and Environment**

### **Introduction**

ACE is a system for computer-assisted animation. It is based on the traditional cel-oriented animation process with the computer taking over many of the tedious jobs. You enter a succession of frames which represent a *sequence*. The system then plays back your frames to create the animated effect. It lets you draw pictures, enter text, and edit your work. The animated images you make are displayed on the screen in real-time. The two main parts of ACE system are a frame compiler and an environment. The environment provides the editing tools, frame manipulation, and display capabilities. The compiler operates automatically to produce a compressed-storage representation for frames.

You can also use the graphic editing features in ACE to make individual pictures, whether or not they're intended to be used for animation. Finally, you can use the compiler directly to compress any bitmap image so that it take up less space on your disk.

The majority of the code for ACE was originally written by Paul Turner, a student at the University of Rochester. I am currently maintaining the system. Please send all bug reports, comments, and suggestions directly to me, Denber.WBST, or Denber.WBST@Xerox.COM (Arpanet). This document describes the features available in ACE version 2.1.

### **Background**

In this document: *holding* the mouse on a menu selection means to press down a mouse button on a menu item (inverting the item) and keeping it down for about 1.5 seconds (at this point, you can release the button or move to another selection). *Clicking* the mouse means pressing a mouse button down and releasing it. Unless otherwise stated, the left mouse button is used for selecting items from menus and to click at objects.

In addition to the mouse, ACE supports a graphics tablet (*Summagraphics MM1201*). [LMM: The graphics tablet hasn't been tested in Medley.] The tablet is more convenient for doing free-hand drawing; in fact, most commercial animation systems include a graphics tablet. The pen has two buttons: the stylus *tip* (which is activated by pressing down on it) and a blue button on the *barrel* of the pen (activated by pressing with the forefinger). We have adopted a convention with regard to the tablet: the stylus button acts like the left button on the mouse and the barrel button acts like the middle button on the mouse.



## Terms and System Organization

A *region* is simply a rectangular area.

A *frame* is a region that contains one complete "picture" in an animation sequence; it is a rectangular bitmap with a fixed width and height.

A *sequence* is a collection of frames defining one complete animated segment.

The *current frame* is the frame in a sequence to which operations will be applied. As all the frames in any given sequence are the same size, you may say that one characteristic of a sequence is a region of a particular size. Frames are referred to by number for convenience; the numbering is from 1 to n.

There are two principal windows used in ACE. The *sequence window* is the window on the screen where a particular sequence will be created, edited and displayed. Typically, you define the shape of the sequence window to give you just the area you want to work in, although a sequence can also be edited and displayed in any existing window.

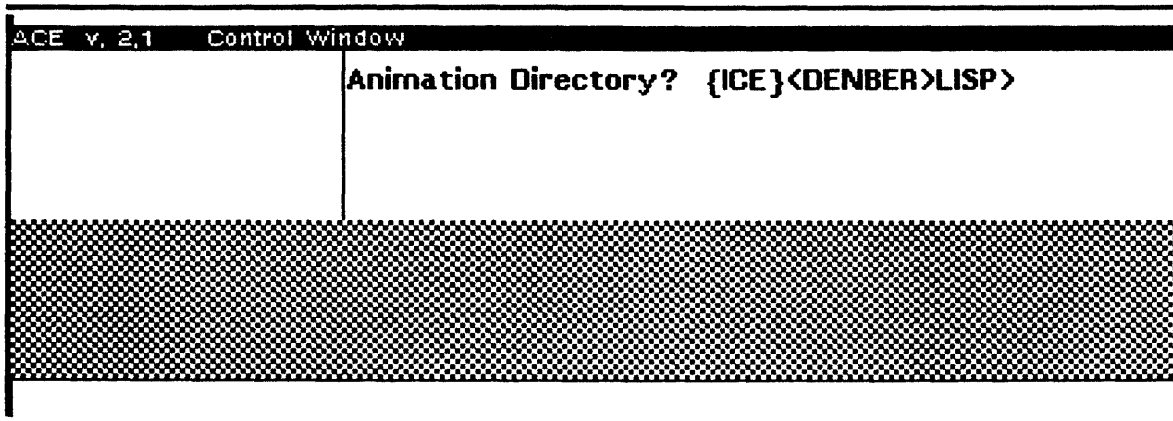
The *ACE Control Window* holds a menu of commands and displays animation state, prompt, and help information. The upper left region in the control window, referred to as the *status region*, tells which frame is currently being displayed (which frame is the current frame); which device (mouse or tablet) is being used in line art and painting operations; what operation is currently being performed; and, the size of a region (width, height) or the location (x, y) of the cursor within the sequence. The upper right portion of the control window is the *prompt region*; it is used to get user input and display helpful information. The bottom part of the control window is a menu of animation functions.

## GETTING STARTED

Load ACE.LCOM from your lispusers directory (e.g., (FILESLOAD (SYSLOAD) ACE). When this is complete, type:

(ACE)

At this point an Ace control window will appear by the cursor. You can place it wherever you wish on the screen. The window initially contains a prompt "Animation Directory?" asking for a default directory to use for storing and retrieving animation files (The default selection is your login directory. Just press the return key to accept the default.) The control window can be moved around just like any other window. While you never need to "quit" from ACE, if you close the control window with the right mouse button menu it permanently aborts ACE; if you then re-type (ACE), the animation system will be restarted from scratch.



*ACE Control Window at start-up*

**ACE Commands**

**Main Menu**

The menu selections from the control window will now be described; an example of using ACE is given in the next section. The main menu is divided into three columns. The left-most one contains commands that affect the entire sequence, the middle column operates on frames, and the right-most column contains utility commands.

Get Sequence	Edit Frame	Run Sequence
Put Sequence	New Frame	Increment Frame
New Sequence	Delete Frame	Decrement Frame
Reset Sequence	Adjust Timing Delays	Initialize MM1201 Tablet
Change compression %	Change Input Device	Quit

*Sequence commands (left column)*

**Get Sequence** Loads a sequence-file from a file server or local disk. You are prompted for the name of the file; incomplete file names will be completed from the directory given as the default ACE directory. You will then be prompted for a window specification. Unless you want to display the sequence in a particular window, select 'create window automatically'.

**Put Sequence** Saves the current sequence to a file. You will be asked for a file name and given the option to overwrite the existing version of the file (if any) or create a new version.

**New Sequence** Discards the current sequence (if any), and prompts you for a new sequence by requesting a size for the new sequence. Dragging out a rectangular region with the mouse; the exact size of the region is displayed in the status region of the control window. A blank first frame is created, ready for editing.

**Reset Sequence** "Rewinds" the current sequence to the beginning (i.e. there is no current frame and the next frame to be displayed will be the first frame).

**Change compression %** This can be used to change the amount of space compression performed by the animation compiler. For general use, there is no need to ever call this command.

**Frame commands (middle column)**

**Edit Frame** Allows editing on the current frame. Brings up a menu of editing options, described in the next section.

**New Frame** Inserts a new frame after the current frame (ie. before the next frame). The frame editor is then automatically invoked.

**Delete Frame** Deletes the current frame. The current frame is removed and the previous frame become the current frame. The first frame can not be deleted.

**Adjust Timing Delays** Lets you set the amount of time (in milliseconds) that any particular frame is displayed; for example, a delay of 50 on the 5th frame would mean that the 5th frame will be visible for 50 milliseconds before the 6th frame is put up. You can change the entire sequence or a single frame at a time. For individual frame setting, you get a menu of frames and their current delays. Holding down a selection will display that particular frame in the sequence window (this is also a convenient way to rapidly move to an arbitrary frame); if you select a frame you will be prompted for a new delay value. When new frames are created, they always get a default delay time of 0.

**Change Input Device** Lets you select either 'mouse' or 'tablet' from a menu. This sets the device to be used for line art and painting operations. The status (upper left) region of the control window always shows which device is active. All menu selections have to be done with the mouse, even when the tablet is being used for drawing

#### *Utility commands (right column)*

**Run Sequence** Runs the *remainder* of the sequence. To run the entire sequence, select Reset Sequence before Run. This command has a submenu with two additional commands:

**Loop** Runs the entire sequence in a continuous loop. To stop the loop, hold down the space bar. This is checked only at the end of the sequence, so just tapping the space bar may not stop the loop.

**Loop part** Runs a portion of the sequence in a continuous loop. You can specify the starting and ending frame numbers. To stop the loop, hold down the space bar, as in Loop above.

**Increment Frame** Displays the next frame and makes it the current frame.

**Decrement Frame** Goes back to the preceeding frame and makes it the current frame.

**Initialize MM1201 Tablet** This performs the necessary RS232 port initializing, sets the baud rate, activates the tablet, etc. This must be called after the tablet is plugged in and before it is used. If your tablet doesn't seem to be responding, it may need to be reinitialized.

#### *Edit Frame Submenu*

For the commands described below, it is sometimes useful to know the exact coordinates at which a drawing operation will take place. If you hold the T key down, ACE will put the current coordinates in the status window. Release the key to stop this function.

When you select **Edit Frame**, a sub-menu of editing options appears. Their functions are as follows:

**Paint** This lets you to paint on and erase bits from the current frame. The painting operation is the standard Interlisp-D window paint command. Either the mouse or tablet can be used (which ever device is currently selected). Pressing the left mouse button or pen stylus draws; the middle mouse or pen barrel button erases. Pressing the left shift key brings up a menu. From this menu, you can

quit painting, change the brush size or shape, and the color or texture of the "paint brush". Note: selecting items from this menu requires using the mouse (unfortunately, the tablet cannot be used for menu selecting). For more information on the paint command, please see page 19.20 in the Interlisp manual.

**Line Art** This lets you add straight lines to a frame by selecting one vertex, dragging out a line, and then selecting another vertex. In this way, an arbitrary string of connected line segments can be created. The left mouse button or pen stylus will "put down" vertices, the middle mouse button or pen barrel stops the line dragging. The right mouse button brings up a menu of line art options (paint or invert drawing; several line width choices); as with all menus, the menu selection must be made with the mouse.

**Edit Bits** This lets you use the Interlisp-D bitmap editor on the selected frame. The mouse is used to turn specific bits on or off (the tablet is not used as it isn't helpful for this application). A complete description of the the bitmap editor is given in the Interlisp Reference Manual. Always exit the editor by selecting **OK**.

**Text** Lets you put text into a frame. After selecting the 'Text' option, you will be prompted for font characteristics. Then you point (with the mouse) to where the text should begin and click the left mouse button. You may now type in text from the keyboard and it will show up in the frame. A press of the return key ends text entering (the return is NOT included in the text).

**Move Region** Lets you move an arbitrary rectangular region in the current frame. You first drag out the region to be moved, then you will be asked what to do with the old image (i.e. leave it alone, erase it). At this point, a ghost image will attached to the cursor and you can position the image in its new location. Clicking the mouse will set the position for the image and then you will be asked how to combine the image (i.e. paint it in, exclusive-or it in).

**Combine Region** This is similar to move region, except that you can select any region on the screen (not just inside the current frame). After selecting the region, bringing the mouse within the sequence window will show a ghost image; the rest of the procedure is the same as for **Move Region**. This command is extremely useful for bringing images created elsewhere into your frame. For example, you might have a drawing made using Sketch or an AIS file.

**Texture Area Fill** This lets you fill in an arbitrary closed curve with a texture pattern. You are first prompted to select a bounding region. This reepresents a maximum area beyond which texturing will not occur, in the event that the texture "spills" outside the region being shaded. Next, select a starting point anywher inside the desired region. You will then be offered a menu of predefined textures. You can choose one of these, or create your own by selecting \* **Other** \*. The area is then filled with that texture. You can then confirm that the right thing happened by clicking left. Click any other button to undo the operation.

**Texture Region Fill** This is like Texture Area Fill, except that it is used to create filled-in rectangular boxes, rather than arbitrary areas.

**Scale Region** Lets you change the size of any rectangular area. You first select a region, and then indicate the shape of the scaled area by sweeping it out on the frame. Useful if you know how big you want the result to look but not what percent of the original it is. This command has a submenu:

**To a new region** Same as the top level Scale Region.

**In x and y** You first select a region as in **To a new region**, and then indicate the percentage of the original size to scale by, much like selecting reduction or enlargement on a copier. You can set the x and y scale factors independently.

**In x only** Use this if you only want to scale in the x direction, leaving y unchanged.

**In y only** Use this if you only want to scale in the y direction, leaving x unchanged.

**Clear Region** For clearing regions to white quickly. If you select a region within the sequence, it is immediately erased. There is no UNDO for this operation.

**Quit - Compile** This is the usual way of exiting the editor. This keeps the changes made to the frame and calls the compiler, resulting in adding the frame to the current sequence.

**Quit - ABORT** Exits the editor but does not update the frame. The sequence will be as it was before you selected **Edit**. Any changes made to this frame are lost.

## CONCLUSION

### Examples

You might want to see an existing animation before creating one of your own. There are several animation demos included in the Lispusers distribution of ACE: ACE-APPLEDEMO, ACE-BOUNCINGBALL, AND ACE-FOUETTE. The simplest one, BOUNCINGBALL, contains five frames showing a bouncing ball. To see this, start ACE running and select **Get Sequence** from the main menu. Type the name of the file, including the file server and directory if they are different from your current animation directory. Once the file is loaded ACE will let you position the sequence window. Then select **Run Sequence**. The system will display the five frames and then stop. To see the ball bounce continuously, select **LOOP** from the submenu on the **Run Sequence** command. The ball will now bounce until you hold the space bar down.

The file ACE-APPLEDEMO is a 125 frame sequence which shows an apple getting shot. ACE-FOUETTE is a six frame cycle of a ballet dancer performing a fouette turn.

### Animation Hints

Remember that ACE is just a tool - it will not do any animating for you. Our goals were to provide a system that simplified the frame creation process and let you create animation on the computer without having to learn a special animation programming language. However, animation is an art form in itself. Experience is gained only through practice and experimentation.

Avoid moving objects too far between frames or the motion will appear jerky. The D machines screens are designed to minimize flicker through the use of a long persistence phosphor. Unfortunately, this results in trailing streaks of light behind rapidly moving objects. Sometimes you can use this to artistic effect. It can be reduced by moving dark objects over a light background, rather than the reverse.

Sometimes you can simplify the animation process by creating frames out of order, especially for cyclic animation. For example, the bouncing ball was created by drawing frame 1 and then making a new frame (which is by default a copy of frame 1). Then we backed up to frame 1 and added a new frame between 1 and 2, showing the ball half-way down. Then this frame was copied, yielding four frames. Backing up again and adding the middle frame gave a symmetrical bounce sequence with frames 1, 5 and 2, 4 being identical.

It's often convenient to keep a snapshot of the object you're animating handy in a window next to the sequence window. It can then be brought into the frame whenever needed, for example in the case where it is being modified in some way. You are not limited to editing images with the ACE editor. In particular, you may want to use the *Sketch* editor (an Interlisp-D library package) to modify images, and then bring them into the sequence window for compilation.

#### References

- Denber, Michel, Paul Turner, "A differential compiler for computer animation", to appear in *Computer Graphics*, 20:3, 1986 (Proc. SIGGRAPH '86)

Fox, David, Mitchell Waite, *Computer Animation Primer*, McGraw-Hill, N.Y. 1984

Magenat-Thalmann, Nadia, Daniel Thalmann, *Computer Animation: Theory and Practice*, Springer-Verlag, Tokyo, 1985

---

---

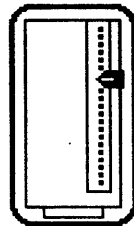
**ADDRESSBOOK**

---

---

By: `dgb (Bobrow.pa@Xerox.com)`

Requires: LOOKUPINFILES



### INTRODUCTION

The ADDRESSBOOK package provides quick and easy access to on-line address books or phone directories. It allows you to copy (shift select) from entries found in the book, for example, for use as a letter or electronic mail address. When you load the ADDRESSBOOK package, the icon shown above will appear on your screen. Opening this icon will provide a window interface to a simple search process. To find an entry containing any string in one of your \*AddressBookFiles\*, type the string followed by a return. The ADDRESSBOOK program will quickly search through the files and show you an occurrence of the string typed. The located string is shown in inverse video. The title of the window will contain the name of the file in which the entry was found. You can use a name, part of the address or any keywords to locate the appropriate part of the text. The search ignores case; e.g. "bobrow" matches "Bobrow". The text of the document is scrollable, and any portion can be shift selected into another document.

If the portion of the file found was not the one desired, either type carriage return or click on Next Occurrence to search further in the files for the same string. If no (further) occurrences are found, the text window will display a message indicating the failure. Clicking Next Occurrence again after failure will restart the search from the beginning of all the files, using the same lookup string. Typing a new string can be repeated as many times as you like. When you are done, just SHRINK the window back to its icon.

---

```

Lookup String: stefik
Lookup String:
Next Occurrence
Looking in: {phylum}<bobrow>lisp>addresses.ted
Dr. Mark J. Stefik
Xerox Palo Alto Research Ctr.
Knowledge Systems Area
3333 Coyote Hill Road
Palo Alto, CA 94304
Residence:

```

---

*Example ADDRESSBOOK window*

## REQUIRED FILES

This package automatically loads LOOKUPINFILES.

## VARIABLES

**\*AddressBookFiles\*** [Variable]

**\*AddressBookFiles\*** is a list of files that contain entries to be searched. This is usually set in the INIT.LISP file. In the ADDRESSBOOK package it is initially set to PHONELISTFILES, to make it backwards compatible with PHONE-DIRECTORY. The **\*AddressBookFiles\*** can be any unformatted or TEDIT formatted files, with any number of lines per entry. A typical value for PARC users (as defined for PHONELISTFILES in PARC-INIT) is

```
( {INDIGO}<REGISTRAR>PARCPHONELIST.TXT
  {INDIGO}<REGISTRAR>ISDNORTHPHONELIST.TXT ).
```

**\*Address-Book-Pos\*** [Variable]

**\*Address-Book-Pos\*** is the initial POSITION for the ADDRESSBOOK icon. This is defined as an INITVAR in the file, so you can set it before loading the file. The default value is

```
(create POSITION XCOORD ← 970
  YCOORD ← (DIFFERENCE SCREENHEIGHT 90)).
```

This places the icon in the upper right corner of the screen.

**\*Address-Book-Region\*** [Variable]

**\*Address-Book-Region\*** is the initial REGION for the ADDRESSBOOK window. This is defined as an INITVAR in the file, so you can set it before loading the file. The default value is

```
(CREATEREGION 300 (DIFFERENCE SCREENHEIGHT 500) 400 200).
```

This places the window in the middle of the screen.



**NOTES****Caching Files**

When you first open the ADDRESSBOOK window, the program will copy the \*AddressBookFiles\* to {CORE}, significantly speeding up queries. Bugging in the title of the ADDRESSBOOK window with the left or middle mouse button will produce a menu with an option to recache the files on \*AddressBookFiles\*.

**Editing Your Files**

To edit the file in which an entry is found, bug in the title of the ADDRESSBOOK window, and select the option "Edit File". You will be requested to confirm that you want to edit the file. If you confirm, a TEDIT process editing the file will be set up. This process is independent of the lookup process. To make editing changes visible to the lookup process, PUT the file in TEDIT; when it is done, recache the \*AddressBookFiles\* as specified above. To recache just the file edited, (the one specified in the title bar of the window), select the option "Recache just this file" in the title bar menu.

**Adding to the List of Files**

To add to the list of files being used for lookup, select the option "Add new file" in the title bar menu. This file will be added, and cached in core.

---



---

## AIREGIONS

### (Active Irregular Regions)

---



---

By: Greg Wexler (Wexler.pasa@Xerox)

and

By: Jim Wogulis (Wogulis@ICS.UCI.EDU)

New Owner: James Turner (Turner.Lexington@Xerox.com)

Uses: FILLREGION, AIREGIONS-DEMO

### INTRODUCTION

The purpose of this package is to provide menu-like operations on irregularly shaped regions within a window and make available general functions that allow users to create their own applications using irregularly shaped active regions. An added feature of AIRegions is that multiple IREGIONS may be activated by selecting the intersecting area of those IREGIONS. (Throughout this document an irregularly shaped region will be referred to as an IREGION).

### DESCRIPTION

Virtually all of the features of menu selection have been implemented in this package: ease of menu creation, item-selected shading, quick response to selection, and execution of an associated function. Yet, this package adds one additional feature without any degradation to the quality and efficiency of menu implementation: the selection of any irregularly shaped region from any point within that region, and without any unsightly cosmetic change.

In describing the package by means of an example, picture a map of the world, or better yet, of a particular country broken up into its individual states and/or provinces. Suffice it to say that these regions are not square but irregular in shape and that they are bordered by solid lines, as they are on a common map. Unlike the menu package or ACTIVEREGIONS package, AIRegions allows you to select any of these pre-set states/provinces just as if you are making a menu selection of an item. One of the nice aspects of this package lies in the fact that the package does NOT make any cosmetic changes to the irregularly shaped region, like providing some small box within the region to button in. Simply button your mouse within the solidly bordered region, anywhere in the region, and it will shade it to your particular shade and execute your defined function.

#### Functionality provided:

The functions in this package allow the user to work with familiar concepts: creating and implementing windows and menus. The examples provided within this documentation should be sufficient for the user to begin setting up irregularly shaped regions.

(CREATEIR *window shade buttoneventfn helpstring region poslist*) [Function]

*window*: the window which will contain the irregular region.

*shade*: can be either a number between 0 and 65535 for a 4 by 4 shading or a 16 by 16 bitmap (if *shade* is NIL then the default is black, 65535).

**buttoneventfn:** the function called when the region is selected. The arguments that are passed to the function are: the window containing the IREGION, the IREGION record itself, and the button which selected the IREGION.

**helpstring:** the string that is placed in the PROMPTWINDOW when the mouse is held over the item for a few seconds.

**region:** if specified, will be the region relative to *window* in which the IREGION can be found. (If *region* is NIL, the user will be prompted to sweep out a region within *window*.)

**poslist:** If specified, will be either a position or list of positions relative to *window* that are the starting points for the FILLREGION routine (i.e. a point within the desired IREGION). (if *poslist* is NIL, the user will be prompted for a position until he/she selects outside of *region*.)

**Description of use:** This is the first function that is called when actually setting up an irregularly shaped region to become sensitive to button activity. If the *region* argument is not set, then the cursor changes its shape and prompts for a region to completely surround the IREGION within the desired active window. (Note: That it is best to surround the desired IREGION as close as possible since this will save on execution time and memory useage.) A thin box will appear temporarily where the IREGION was scanned. If *poslist* is NIL, then the cursor changes into a TARGET symbol. The user should left-button mouse within desired active IREGION. Note: the IREGION must be surrounded by a border that FILLREGION can use to define the active area. Any gaps in the IREGION will cause the next routine to fill the region and anything outside with the shade provided. Mistakes can be corrected by using the REMOVE.IREGION function described below and PAINTing in the gap to retry. After left-buttoning within the desired active IREGION, the cursor continues to remain in its TARGET state. If the IREGION is split up into many different parts, those parts may be selected with the left-button also making them all active concurrently. However, when one is finished activating that one IREGION, then she/he should left-button outside of *region*. This function must be called for each desired IREGION.

**Examples:**

```
(CREATEIR window 21930 'myfunction "This is the helpstring")
```

```
(CREATEIR (WHICHW) 1234 'MY.SELECTED.FN "This is the helpstring"
 '(0 0 20 30) '((12 . 15)(2 . 29))
```

```
(SURROUNDIR window shade buttoneventfn helpstring poslist inside.pos) [Function]
```

**window:** the window which will contain the irregular region.

**shade:** can be either a number between 0 and 65535 for a 4 by 4 shading or a 16 by 16 bitmap (if *shade* is NIL then the default is black, 65535).

**buttoneventfn:** the function called when the region is selected. The arguments that are passed to the function are: the window containing the IREGION, the IREGION record itself, and the button which selected the IREGION.

**helpstring:** the string that is placed in the PROMPTWINDOW when the mouse is held over the item for a few seconds.

**poslist:** If specified, a list of positions relative to *window* that are the edge points for the FILLREGION routine. If NIL, the user will be prompted to define the outer border of the region desired to be active. Holding the SHIFT key will define the last point used in defining the edge. If this field is non-nil, *Inside.pos* must be specified.

*Inside.pos*: If specified, this would be the inside position in which the Fillregion routine would begin filling from. If *poslist* is non-nil, then this field must be specified.

**Description of use:** Like the CREATEIR function, this function creates IREGIONS. However, the functionality of this routine is quite different. There are times when you do not care what is within a particular region. Say, for example, you have a map of some country and you wish to surround a particular region of the country with an IREGION as you wish to denote an area rich in some mineral deposit or some other characteristic. Such a characteristic is oblivious of the borders of the country's states or provinces, streams, rivers, etc., yet you would like to make active a very general area. Upon calling this function, you are prompted to button around the area of interest. And so, in viewing the crosshairs cursor, you begin buttoning about specifying the border of the area you wish to make active, independent of what is inside it. To stop being prompted for the next edge, simply hold the SHIFT key on the keyboard, (either one will do), as you make your last button selection. At this point, the lisp DRAWCURVE function will take effect and draw the closed region you've defined. Note that the first and last points do not have to touch as the DRAWCURVE routine will connect them for you. You will also be prompted to button within the region you've marked. It is here that the Fillregion routine will begin filling your region from. When complete, this function adds the IREGION to the window and returns the iregion added.

**Examples:**

```
(SURROUNDIR window 21930 'myfunction "This is the helpstring")
```

```
(SURROUNDIR (WHICHW) 1234 'MY.SELECTED.FN "This is the helpstring"
 '((5 . 5)(6 . 50)(50 . 50)(50 . 7)) '(10 . 10))
```

(ADD.IREGION *window iregion*)

[Function]

*window*: the window to which the iregion is to be added.

*iregion*: the IREGION to be added to window.

**Description:** This function will add iregion to window which will then allow mouse selection of that IREGION.

(REMOVE.IREGION *window iregion*)

[Function]

*window*: the window in which the *iregion* exists.

*iregion*: the IREGION you wish to remove from *window*.

**Description:** This function removes the region from a list of active irregular regions which is stored as a window property of the window. The list of irregular active regions can be found by evaluating: (ALL.IREGIONS *window*).

(INTERSECTING.IREGIONS? *window flg*)

[Function]

*window*: a window.

*flg*: either T or NIL

**Description:** This function sets up window to allow selection of intersecting iregions. If two or more iregions overlap and this function had been called with *flg* = T, then when the overlapping region is selected, all of those iregions will be high-lighted and each IREGIONS.BUTTONEVENTFN will be called. If *flg* is set to NIL, then the last IREGION created in that intersection of iregions will be selected. (Please be aware that intersecting iregions *might*

*generate effects that you do not wish to have.* That is, if you leave the iregion "ON" (the exact same thing you see when you hold the mouse button down on the iregion, done by inverting that iregion) and create another iregion intersecting with the first, then the mask of the second would have a partial image of the first. At this point, buttoning in an area where both regions intersect might show everything but the intersection of those regions. Sometimes, it all depends on the *order* that they are created and what iregion's mask is left on or off. Shades that are "negatives" or "equals" of one another might make matters more complex than necessary when they are intersected. It is recommend that you play with this function in order to understand how it actually works so that when you work it into your application you'll have a better idea of the functionality and end-results). If this becomes a problem, an EDIT.MASK function has been provided so that you may edit the mask of the iregion by hand. Currently, there are no programmatic methods for doing this.

(ALL.IREGIONS *window*) (Function)

*window*: a window containing IREGIONS.

**Description:** This function returns a list of all the IREGIONS attached to *window*.

(DOSELECTED.IREGION *window iregion button*) (Function)

*window*: the window associated with iregion.

*iregion*: the iregion to be activated

*button*: the button which selected iregion.

**Description:** Applied iregions BUTTONEVENTFN to window, iregion and button. This provides a programmatic way of activating a given IREGION. This does not invert the iregion.

(EDIT.MASK *iregion*) (Function)

*iregion*: the IREGION whose mask you want to edit.

**Description:** This function is provided for buttoning in places where the MASK is not set. More explicitly, TARGETing a region (while creating the regions) specifies the places where the FILLREGION routine is to create a mask. For example, if a US state contains many rivers one pixel wide, the FILLREGION routine will fill around the river, but not the river itself. This means that when the mouse is positioned on the river, the region will not shade because the mask does not have that bit turned on. However, if the mask is edited and the rivers filled in, buttoning on those rivers will activate the IREGION.

(INVERT.IREGION *window iregion*) (Function)

*window*: the window in which the *iregion* exists.

*iregion*: the IREGION targeted for shading.

**Description:** This will highlight the *iregion* with that *iregions* shade. Calling it a second time will low-light it.

(IREGIONP *iregion*) (Function)

*iregion*: the IREGION to be tested.

**Description:** This function returns NIL if *iregion* is not an IREGION datatype and returns *iregion* if it is an IREGION.

(IREGIONPROP *iregion prop newvalue*) (Function)

*iregion*: the region of which you are setting/requesting the property.

*prop*: the property in which you are interested.

*newvalue*: the new value to be assigned to *prop*.

**Description:** As with WINDOWPROP, if *newvalue* is not specified, it will return the current value of the *iregion*'s property. If *newvalue* is specified, then the property will be reassigned with that value. If a *prop* name is not one of the fields of an IREGION record, it will be stored in property-list format on the USERDATA field of the *iregion* record.

**IREGION fields:**

BUTTONEVENTFN - function called when *iregion* is selected.

USERDATA - property list format for user properties (similar to WINDOWPROP).

REGION - region relative to the window that surrounds the *iregion*.

MASK - a bitmap the same size of REGION that is blackened where the *iregion* is active.

SHADE - the shade number or bitmap used to shade the region.

HELPSTRING - the string that is printed in the *PROMPTWINDOW* when a region is held.

**Examples:**

(IREGIONPROP *iregion* 'SHADE) -- returns shade of *iregion*

(IREGIONPROP *iregion* 'SHADE 21930) - assigns new shade to *iregion*.

(SHOW.ALL.IREGIONS *window shade delay*) (Function)

*window*: the window in which the IREGIONS exist.

*shade*: the shade with which the iregions will be shown.

*delay*: the time (in milliseconds) between which each IREGION is displayed . (if *delay* is NIL, then a default of 500 is used.)

**Description:** This function will shade and unshade in *shade* (black is used if *shade* is NIL), each IREGION that has been created in the particular window. This is especially useful when the user has lost track of the number of IREGIONS within a window.

(WHICH.IREGIONS *window posorx y*) (Function)

*window*: the window in which the IREGIONS lie. (if *window* is NIL, default is window to which mouse points).

*posorx, y*: the location within the window where the IREGIONS can be found. These points must be local to the window's coordinates...not the screen. (if *posorx* is a position, then it will be used, otherwise if *x* or *y* are not numbers then the current mouse position is used.)

**Description:** Will return either NIL or the list of IREGIONS found in *window* and specified by *posorx, y*.

**Examples:**

```
(WHICH.IREGIONS)
(WHICH.IREGIONS MY.WINDOW 50 23)
(WHICH.IREGIONS MY.WINDOW '(50 . 23))
```

**Saving IRegions**

IREGIONS can be saved on a file by setting a variable to be the value returned by ALL.IREGIONS. This variable can be saved by using the file package command, UGLYVARS.

**Example:**

```
(SETQ IRS (ALL.IREGIONS (WHICHW)))
(SETQ SAVEIRSCOMS '((UGLYVARS IRS)))
(MAKEFILE 'SAVEIRS)
```

The file SAVEIRS can be loaded and IRS will be set. You can then add IRS to a window by doing:

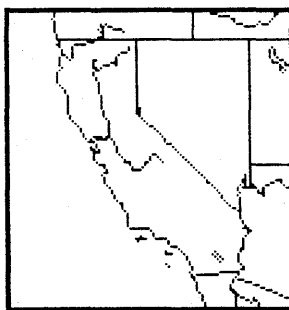
```
(WINDOWPROP (WHICHW) 'IREGIONSLIST IRS)
(WINDOWPROP (WHICHW) 'BUTTONEVENTFN 'IN.CURSOR.REGION)
```

Caution: Some properties on the USERDATA field of an IREGION might not be saved correctly such as a window which can not be saved on a file.

Window images can be saved on a file by creating a bitmap the same size as the window, BITBLT from the window to the bitmap, and then saving the bitmap with the file package command VARS.

**Example use of the AIRegions package:**

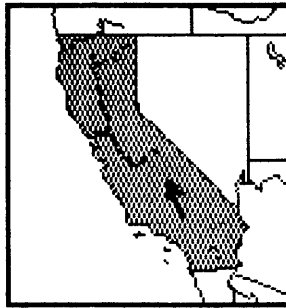
1. Open a window...about 1/4 of a screen.
2. Use the paint function provided when you right-button in the window and paint a picture.



3. With your mouse in this painted window, type in:
 

```
(CREATEIR (WHICHW) 21930)
```
4. The cursor changes shape and prompts for creating a region similar to the prompt for creating a window. In this case, span a region that contains California.

5. When you are done, and the mouse button is released, the region spanned will remain temporarily on the screen. The cursor changes into a target and now prompts for a left-button within the region. Select somewhere in California. When done, left-button the mouse outside and away from the temporarily blocked off region. (If you want to continue selecting areas of the same irregular region, in this example, the upper left corner of California, then button that area within the squared off region. As you can see, your irregular region does not necessarily have to be connected).
- 6. To test it out, simply button anywhere in California and it will fill to a nice shade of grey, as we have just set it up to do:



7. To create more active irregularly shaped regions, follow steps 3 through 5 above. If you want to set the selection of one of the regions to activate the execution of some function that calls RINGBELLS, and have the region shade to black upon selection, type in the following in the top level typescript window keeping the mouse within the painted window.

```
(CREATEIR (WHICHW) 65535 'IR.TESTFN)

(DEFINEQ (IR.TESTFN (LAMBDA (WINDOW IREGION BUTTON)
  (If (EQ (QUOTE LEFT) BUTTON)
    then (RINGBELLS 2))))))
```

Span the cursor out over another state/region and repeat steps 3-5 above. When you button in this IREGION, the IREGION will temporarily shade black, and call the RINGBELLS function. Note that like menu selection, the function is called only when you release the button within the region. If the mouse button is held down and you move over the created IREGIONS, they will shade and unshade as you enter and exit them.

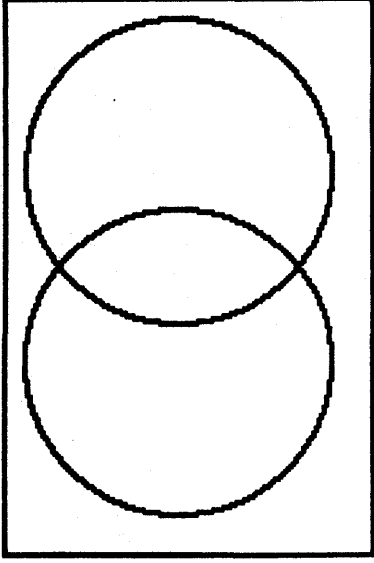
Note: if you wish to create your own shades but don't know what shades correspond to which numbers, call the function (EDITSHADE) and begin selecting points that you want shaded. When you are done, the function will return the appropriate shade number. You can also use 16x16 bitmaps for the shade of an IREGION (try (EDITBM (BITMAPCREATE 16 16)))

DEMO PACKAGE: To run the demo package, load AIRegions-Demo.

### Intersecting Iregions

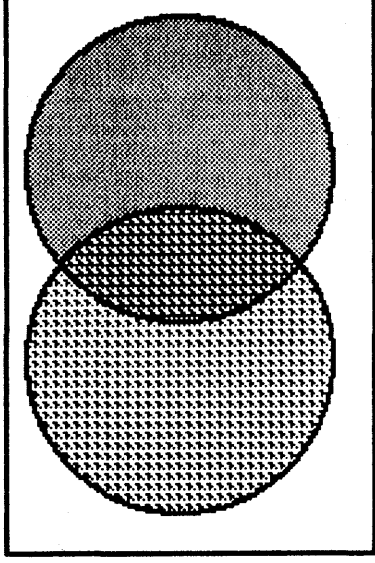
1. Create a window and paint in the following:





2. Now call `CREATEIR` passing in this window and a shade of 4747 and surround the left circle and select inside that circle and also in the intersecting area for the area fill. Repeat this for the right circle but use a different shade (say 42405).

3. Now, with your mouse in the window, call the function `(INTERSECTING.IREGIONS? (WHICHW) T)`. When you button in the intersection of the two circles, you should get:



4. When the mouse is released inside of the intersecting region, both `IREGIONS` `BUTTONEVENTFN` will be called.

Comments and suggestions are welcome.

---

---

**Analyzer**

---

---

By: Maxwell (Maxwell.pa@Xerox)

**-INTRODUCTION**

The Analyzer package is used by the Proofreader (see PROOFREADER). It defines a class of analyzers, of which the proofreader is but one. Later, analyzers will be developed for languages other than English.

---



---

## BACKGROUNDIMAGES

---



---

By: Burwell (Burwell.pa@Xerox.com)

### Accessory files:

Background-DurerCat.bitmap  
 Background-Parc.press  
 Background-Rhine.press  
 Background-Steinheim.press  
 Background-TwoDollar.press  
 BackgroundMenu.dfasl  
 BitMapFns.lcom

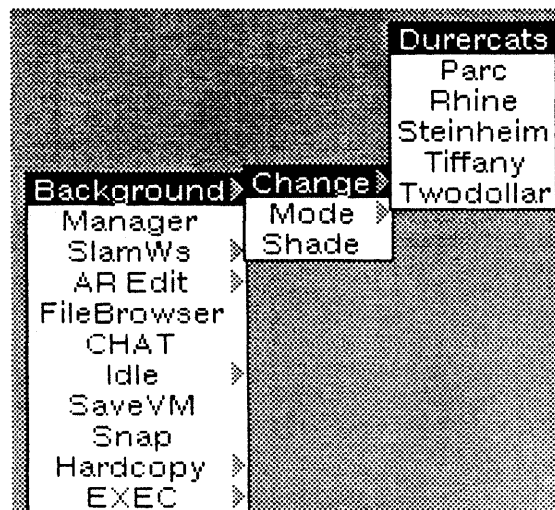
This document last edited on September 8, 1988.

### INTRODUCTION

BackgroundImages is a module which makes it easy to apply graphically interesting static images to the background of one's Lisp screen. To use the package in the simplest way, load it and call

(BACKGROUND.SETUP) [Function]

This will put an entry called "Background" on your background menu (in a manner compatible with the module BACKGROUNDMENU), so that it will look something like this:



If (as shown) you select one of the subitems of "Background>Change," your background will be painted with the image whose name you selected. The background images currently available are

DurerCats: a reflected picture of a cat from an engraving by Albrecht Durer  
 Parc: a picture of the Xerox Palo Alto Research Center

Rhine: a picture of a village on the Rhine river  
 Steinheim: a picture of a relatively unfortified castle  
 TwoDollar: a picture of part of a two dollar bill

If the image you select is a different size than your screen, you may want to control how the image is applied. There are three different image painting modes: "Center," which centers the image on the screen and paints gray in the remaining space; "Tile," which tiles the screen with the image; and "Reflect," which tiles the screen with edge-matched reflections of the image. (This last mode is particularly effective with the DurerCats image.) To change the mode, select one of the subitems of "Background>Mode." To the currently set mode, just select "Background>Mode" itself. Note that once you have changed the mode, to see its effect, you must reapply the background image (by selecting "Background>Change>ImageName").

If you want a less busy background, you can use a plain gray. To apply it, just select "Background." To change the shade of gray, select "Background>Shade." Again, to see the effect of the shade change, you must reapply the background shade (by selecting "Background").

#### DETAILS

Background images to be used with this module must be represented in files that can be read by either HREAD or READPRESS. For convenience, they should be named according to the conventions mentioned below under BACKGROUND.FILES.

BackgroundImages does take some pains to reduce user wait time. First, it is very lazy about file interactions, and defers them until it is quite clear they cannot be avoided. And second, when one selects a background, it is cached so that changing back to it will be significantly faster than fetching it the first time. Since the cached background bitmaps consume quite a bit of space, they can be removed by the GAINSPACE mechanism.

The public interface to this package, more fully described, is as follows:

(BACKGROUND.SETUP NAMES) [Function]

Puts an entry on the background menu which enables users to change backgrounds easily. The entry will be labeled "Backgrounds" and if invoked will turn the screen background gray. The entry will have several subitems, each labeled with the name of the background image it will, if selected, put on the screen. The argument NAMES is meant to specify the names of the images; it must be a list either of dotted pairs (whose CAR is the name of an image and whose CDR is the name of the file in which a representation of that image can be found) or of atoms (each of which is the name of an image). If NAMES is NIL, BACKGROUND.SETUP will call (BACKGROUND.FILES) to generate a set of background image names.

(BACKGROUND.FILES WHICH) [Function]

Returns a list of dotted pairs whose CAR is the name of an image and whose CDR is the name of the file in which a representation of that image can be found. Generates this list by looking on LISPUSERSDIRECTORIES for files of the form "background-\*.bitmap" or "background-\*.press"; all such files are taken to be representations of background images. Image representation files that are not named and located according to this convention will have to be specified directly to BACKGROUND.SETUP. If WHICH is T, it will search all the LISPUSERSDIRECTORIES; otherwise it will search till it finds the first directory with background images in it.

**(BACKGROUND.FETCH *NAME FILENAME MODE*)** [Function]

Causes the image whose name is *NAME*, and for which there is a representation in file *FILENAME*, to be applied to the screen background. It is this function which the background menu subitems call to apply new images. If *FILENAME* is not specified, **BACKGROUND.FETCH** will attempt to find an image representation file whose name is either "background-*NAME*.bitmap" or "background-*NAME*.press" on any of the **LISPUSERSDIRECTORIES**. *MODE* specifies how the image will be applied to the background if it is a different size than the screen. *MODE* should be one of the atoms **CENTER**, **TILE**, or **REFLECT**; it defaults to **CENTER**. **CENTER** causes the image to be centered with a white border around it; **TILE** causes the image to tile the screen; and **REFLECT** causes the image to tile the screen such that each tile is a reflection of those adjacent to it.

**(BACKGROUND.MODE *MODE*)** [Function]

Sets and accesses the mode (as described above) which will be passed to **BACKGROUND.FETCH** when the latter is invoked from the background menu subitems. *MODE*, if provided, gives the new mode setting. Returns the previous mode setting.

**(BACKGROUND.SHADE *NEWSHADE*)** [Function]

Changes the default background shade.

**(BACKGROUND.CENTER *BITMAP*)** [Function]

Returns a screen-sized bitmap with *BITMAP* centered in it with a border colored with the default background shade.

**(BACKGROUND.TILE *BITMAP*)** [Function]

Returns a screen-sized bitmap that is tiled with *BITMAP* with one of the tiles centered.

**(BACKGROUND.REFLECT *BITMAP*)** [Function]

Returns a screen-sized tiled bitmap such that each tile is a reflection of those adjacent to it and such that the center tile is a copy of *BITMAP*.

---



---

**BACKGROUNDMENU**

---

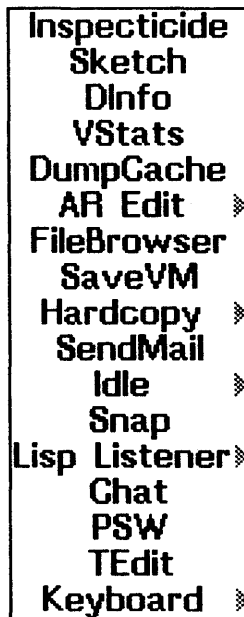


---

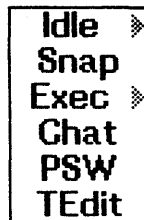
By: Mike Dixon  
 New Owner: Burwell (Burwell.pa@Xerox.com)

**INTRODUCTION**

If you love to load all those fun LispUsers packages but can't deal with background menus that look like this:



don't despair! With just a few quick calls you can have a background menu that looks like this:



(don't worry, they didn't disappear, they're just hiding under "Exec").

**DESCRIPTION**

BackgroundMenu defines several functions for rearranging your background menu to suit your taste.

- (BkgMenu.rename.item *item newname*) [Function]  
changes the name of a background menu entry
- (BkgMenu.move.item *item superitem atend*) [Function]  
makes *item* a subitem of *superitem*. If *atend* it is placed after any subitems of *superitem*; otherwise it is placed before them. If *superitem* is NIL *item* is placed at the top level of the menu.
- (BkgMenu.reorder.items *superitem atend*) [Function]  
just like BkgMenu.move.item but moves a list of items. Useful for changing the order of the items in a menu.
- (BkgMenu.remove.item *item*) [Function]  
throws *item* out of your background menu.
- (BkgMenu.fixup) [Function]  
BackgroundMenuTopLevelItems [Variable]  
BackgroundMenuFixupMode [Variable]  
each top level item which isn't on the global BackgroundMenuTopLevelItems is made a subitem of BackgroundMenuSuperItem. If BackgroundMenuFixupMode is 'top they're added before any subitems of BackgroundMenuSuperItem, if it's 'bottom they're added after, and if it's NIL items moved from the top are added at the top and items moved from the bottom are added to the bottom.
- (BkgMenu.subitems *item*) [Function]  
returns a list of the subitems of *item* (or the top level items, if *item* is NIL).
- (BkgMenu.add.item *item superitem atend*) [Function]  
adds a new menu item *item* as a subitem of *superitem*. If *atend* it is placed after any subitems of *superitem*; otherwise it is placed before them. If *superitem* is NIL *item* is placed at the top level of the menu

**EXAMPLES**

As an example of using BackgroundMenu, this is what I've got in my init file (which produces the changes shown above) (note that i've already loaded LISTEN):

```
(BkgMenu.rename.item "Lisp Listener" " Exec ")
  (* "Lisp Listener" is just too long. the blanks before and after Exec are just there to
  improve the spacing)

(SETQ BackgroundMenuTopLevelItems '(Idle Snap " Exec " Chat PSW
TEdit))

(SETQ BackgroundMenuSuperItem " Exec ")

(BkgMenu.fixup)
  (* Push everything i don't use regularly under the now-renamed Lisp Listener)

(BkgMenu.reorder.items BackgroundMenuTopLevelItems)
  (* and put the top level items in the order i prefer)
```

If I later add more packages which add junk to the top level of my background menu, just calling `(BkgMenu.fixup)` again will hide anything new under " Exec " with the rest of the junk.

When any of the above functions (except `BkgMenu.add.item`) require you to specify an item, you can usually just give a string with the menu entry (or an atom, which is coerced to a string). The case has to be correct, and blanks have to be in the right place. The function will do a breadth first search of the background menu and all its submenus to find such an entry. If for some reason you have the same entry in more than one menu, you'll have to disambiguate it. To do this, you pass a list for the item, where the first thing in the list is the menu entry, and the rest of the list is a path through the tree to find it. For instance, the item (one two three) means find an entry whose text is "three", then find an entry in the tree underneath it whose text is "two", and the find an entry under that whose text is "one".

The item argument to `BkgMenu.add.item` is a standard menu item, i.e. a list of (label form help.string).

All of the functions return T if they were able to do as asked and NIL otherwise (you tried to do something with a menu entry which isn't there, or you tried to make a circular menu structure). The only exception to this rule is `BkgMenu.subitems`, which as previously mentioned returns a list of the subitems, or the atom `NotAnItem` if it's given a nonitem.



---



---

**BITMAPFNS**


---



---

By: Larry Masinter (Masinter.PA@Xerox.COM)

This document last edited on 4-mar-87

(READBINARYBITMAP *WIDTH HEIGHT FILE*) [Function]

reads a series of bytes from *FILE* and creates a *WIDTH* times *HEIGHT* bit map with contents. Note that each scanline of the bit map is rounded up to the nearest multiple of 16 bits (two bytes).

(WRITEBINARY BITMAP *BITMAP FILE*) [Function]

writes out *BITMAP* to *FILE* in format read by READBINARYBITMAP. Please note that READBINARYBITMAP must be supplied with width and height.

(WRITEBM *FILE BITMAP*) [Function]

writes *BITMAP* on *FILE* first preceding with width and height (in binary) such that it can be read in with READBM.

(READBM *FILE*) [Function]

reads width, height, and then appropriate size bit map.

(WRITEBMLST *FILE LST*) [Function]

writes a list of bit maps on *FILE*.

(READBMLST *FILE*) [Function]

reads a list of bit maps.

The following functions open and close *FILE*.

(READPRESS *PRESSFILE*) [Function]

reads press file *PRESSFILE* and returns a bit map. Can only handle press files generated by PRESSBITMAP and a couple of other utilities. Has no smarts, and is not easily extended.

(WINDOWBM *BITMAP POSITION*) [Function]

creates and returns a window containing image of *BITMAP*. Will be at *POSITION* or (GETPOSITION).

---

---

**CALENDAR**

---

---

By: Michel Denber (Denber.WBST @ Xerox.COM)

Uses: TABLEBROWSER, TEDIT

## INTRODUCTION

CALENDAR is a program which can be used to display a calendar on your screen, and keep track and remind you of events and appointments. Calendar 2.04 (the current distributed version) runs in the Koto or Lyric releases of Lisp. The version number appears in the title bar of each Calendar window. Calendar needs the Lisp Library package TABLEBROWSER, which it loads automatically. It also uses TEdit. Various font sizes (from 8 to 36) in the families TimesRoman and Helvetica may be needed, depending on the size chosen for month windows. Reminder files created by earlier versions of Calendar are incompatible with this version.

## I. STARTING CALENDAR

Load CALENDAR.LCOM from your favorite LispUsers directory, eg.

```
LOAD ({ERIS}<LISPUSERS>CALENDAR.LCOM]
```

and then type (CALENDAR). You will get a menu of years (the menu always shows five years starting with last year). If you select a year with Left, it will create a Year window containing a calendar for that year. Each month in the Year window is also a menu item. If you now select a particular month with Left, CALENDAR will create a Month window showing a calendar for that month. You can now select a particular day within the month to bring up a Day browser (described in the next section). The Month window also shows small calendars of last month and next month. You can bring up those months in the current month window by selecting them with Left. If you select them with Middle, the program will create a new window for that month. The Year menu has an entry labelled "Other". If you select this, it will prompt you to type in a year, if you want one that isn't on the menu.

You can have as many year and month windows open at the same time as you like. Month and day windows can also be reshaped to occupy less room on the screen. You can Shrink any of the CALENDAR windows to an appropriate icon, or close them when they are not needed. The reminder facility remains active. If you close your last year window, call (CALENDAR) again to get a new one. CALENDAR uses the Lisp Prompt Window to display informative messages.

Please send your comments, suggestions, and bug reports to me - Denber.WBST (ARPA: Denber.WBST@Xerox.COM). Thanks.

## II. REMINDERS

### The Day Browser

Clicking Left in any day in a month window will open a browser on that day. The browser displays each reminder for the day, along with its event time if it is a timed reminder. You may have more than one browser open at the same time. When you close a month window, it will automatically

close all day browsers for that month. There is a menu across the top of the browser with the following items:

**Add:** Lets you create a new reminder in this day. If you select Add, the program will bring up a TEdit window containing a template for the new reminder. The template contains several fields you can select and fill in. These are described in Creating Reminders, below.

**Display:** Brings up the full contents of the reminder in a TEdit window.

**Delete:** Useful for deleting reminders that you no longer need. By default, timed reminders are deleted automatically after they "fire"; untimed reminders do not fire and are never deleted automatically. Calendar will immediately remove reminders which you delete from the month window (and the reminder's line in the day browser is crossed out), however it will leave reminders that have fired visible in the month window until you redisplay it (eg. September is visible, you select Redisplay from the right-button menu in the title bar or select September again in the Year window, and all fired September reminders will be purged from the month window when it redraws).

**Update:** Saves your reminders to disk (see the section on Saving reminders below).

**Send Mail:** Prompts you for a name to send to. The selected reminder will be mailed to that person when it activates, rather than displaying on your screen. Note that no validity checking is done when you enter a name, so your message could conceivably not be delivered if you typed the name wrong, for example. The message is mailed when the time arrives. Of course, this assumes that your system is running at that time, that you have Lafite active, and that Lafite is running in the mode (GV or NS) corresponding to your intended recipients.

**Period:** Brings up a menu with the choices Daily, Weekly, Monthly. The selected reminder will be made periodic and will appear at the selected intervals.

### Creating Reminders

You can create a new reminder either by clicking Add in a day browser, or by clicking the middle button in a day box in the month window. This opens a new reminder form with the following fields:

**Title:** The reminder title should not exceed one line in length. This field will be displayed in the Day browser and the month window. This field may not be omitted; all others are optional.

**Event time:** The scheduled time for the event. By default, this is also the time at which the reminder will be activated. If this field is omitted, the reminder is "untimed". Untimed reminders do not alert you. When a timed reminder activates, it beeps and brings up a TEdit window containing the full reminder text.

**Alert time:** The time at which you would like the reminder to activate. You might want to be reminded of a meeting 10 minutes early, for example. The alert time can be set to any time, before or after the event time, as long as it is in the same day. If this field is omitted, it defaults to the value of the event time.

**Alert:** Edit this field to contain just the word Yes or No. If you choose No, the reminder will not alert you, even if it is a timed reminder. If this field is omitted, it defaults to the value set in the Options menu (see Programming below).

**Duration:** The expected length of the event. Version 2.04 makes no use of this field.

**Message:** The actual message you want to save. This may be any TEdit text or omitted entirely.

The new reminder form includes a menu with the choices Save and Abort. After filling in the fields you want, clicking Save will add the reminder to the system and close the form. Clicking Abort at any point cancels the reminder being created.

The time can be entered in almost any reasonable format, eg. 9:00 AM, 9 AM, 9 a.m., 2:30 PM, 2:30 P.M., 1430, or can be left out by skipping over the field. Times are "AM" by default, so if you only type 8:30, it will assume 8:30 AM. A heuristic is included to ask "Are you sure?" if you type a time earlier than 9 without an AM/PM qualifier (this value is controlled by CALDAYSTART, see Programming, below). Times of noon and midnight are special cases. There is no generally accepted meaning for the expressions "12:00 AM" and "12:00 PM". If you want a reminder at noon, enter the time as "12:00" or just "1200". Because reminders are added to a particular day, midnight is ambiguous; there is no provision for entering a time of midnight.

If you add a reminder for a time that is already in the past (for example, to keep a historical record of an event after the fact), the program will save the reminder but will warn you that the reminder time has already passed.

Expired timed reminders are automatically deleted upon expiration by default. Setting the variable CALKEEPEXPIREDREMS (see Programming, below) will cause timed reminders to be retained after firing.

Reminders which are scheduled for a time when your machine is not running will not be activated the next time you login. This avoids having a possibly long sequence of "dead" reminders popping up at login time.

### **Saving and loading reminders**

You can save your reminders in a file at any point. The first time you start Calendar, it will ask you to provide a default host and directory for reminder files. You should enter this in the usual format, for example {DSK}<Lispfiles> or {ERIS}<your-name>LISP>. This will become the new value of CALDEFAULTHOST&DIR (it is initially NIL). To save your reminders, select Update from any day browser. This will open a pop-up menu of currently loaded files, plus an "other" item for giving a new file name. If you enter a new name, all currently unsaved reminders will be stored under that name. If you select an existing file, the contents of that file will be updated and any new reminders created since the last update will be added to it. If you abandon your sysout or if your machine crashes, you can have Calendar automatically reload your reminders file when you restart (see CALDEFAULTHOST&DIR and CALLOADFILE in Programming, below). You can also load a reminder file at any time by holding the middle button down in the title bar of a month window. This will open a pop-up menu of files that have already been loaded, plus an "other" item to specify a new file. In this version of Calendar there is never any need to load a reminder file more than once. The menu is useful, however, to show which files have already been loaded.

An "almanac" reminder file is distributed along with Calendar. It contains a variety of holidays and notable dates for the year. The file is called CALMANACnn, where nn is the last two digits of the year. For example, the file for 1986 is called CALMANAC86. You can load this file by selecting Other from the middle button menu and typing CALMANAC86.

By default, the program will only save your reminders when you select Update. You may control file updating by changing the Auto File Update option available under the Options menu item in the month window. See Programming, below.

### III. PROGRAMMING

A programmatic interface is provided to let you create day, month, or year windows from your own programs.

If your reminder text is a Lisp list (anything inside parentheses), when the reminder fires the program will evaluate the list rather than displaying the reminder in a window and beeping.

#### Functions

(CALENDAR *m d yr*) [Function]

*m*, *d*, and *yr* are integers specifying a month, day, and year, respectively. Arguments are specified as follows:

If only *yr* (must be 4 digits) is supplied, brings up a year window for that year and returns *yr*.

If *m* and *yr* are supplied, brings up a month window for that month and returns *m*.

If *m*, *d*, and *yr* are supplied, brings up a day window for that day and returns *d*.

For invalid combinations (missing *yr*, *d* and *yr* only), returns NIL. Also returns NIL if *yr* is out of range (the calendar algorithm is only valid for years between 1700 and 2100).

#### Examples:

(CALENDAR NIL NIL 1984) shows a calendar for 1984 and returns 1984.

(CALENDAR 10 NIL 1984) shows a calendar for October 1984 and returns 10.

(CALENDAR 10 NIL 84) returns NIL (out of range).

(CALENDAR 10 21 1984) shows October 21st, 1984 and returns 21.

You can also call Calendar with the keywords TODAY, THISMONTH, and THISYEAR.

#### Examples:

(CALENDAR 'THISYEAR) shows a Year window for 1986, if this year is 1986. This might be used in an init file, to always start a Calendar of "this year".

(CALENDAR 'TODAY) opens a Day browser for today, containing all of today's active reminders.

(CALLOADFILE *file-name*) [Function]

Loads the file *file-name* into the reminder system and returns T. Returns NIL if the file is not found or is not a valid reminder file.

#### Example:

(CALLOADFILE '{DSK}<LISPPFILES>CALREMINDERS)

#### Variables

CALALERTFLG [Variable]

Initially T. This controls whether or not reminders whose Alert field is not specified should alert you when they fire. T means they will. NIL means they won't.

**CALDAYDEFAULTREGION** [Variable]

Initially (32 200 350 100). This specifies the default size for day browsers. The location is only used for day browsers opened programatically.

**CALDAYSTART** [Variable]

Initially 900. This represents the time (in 24 hour format) at which your regular day starts. The system will use it to confirm times you enter without a "PM" indicator if they are less than this value. For example, it is more likely that 4 means 4 PM than 4 AM.

**CALDEFAULTALERTDELTA** [Variable]

Initially 0. This represents the time (in minutes) before or after the event time you want reminders to be activated, if no explicit alert time was given for them. To be reminded before the event, make this value negative. The resulting time must still be in the same day as the event.

**CALDEFAULTHOST&DIR** [Variable]

Initially NIL. This is the host and directory on which your reminder files will be saved if you type the file name without a directory specification. The system will prompt you to enter a value for this the first time you start it.

**CALFLASHTIMES** [Variable]

Initially 0. Specifies the number of times to flash the destination given by CALFLASHTYPE when a reminder is activated.

**CALFLASHTYPE** [Variable]

Initially 'None. Specifies which window should be flashed when a reminder is activated. Can be set to 'WINDOW, to flash the reminder display window, or 'SCREEN to flash the entire screen. CALFLASHTIMES (above) should be set to the desired number of flashes.

**CALFONT** [Variable]

Initially 'TimesRoman36. This variable controls the font used to display the Month Window. You can change it for example, by saying (SETQ CALFONT (FONTCREATE 'HELVETICA 18)). The change takes effect the next time you display a month. If you reshape a month window, the program will try to find a smaller font to fit the new window size, but the value of CALFONT will not be changed.

**CALHARDCOPYPOMFLG** [Variable]

Initially T. This variable controls the printing of the phase-of-the-moon icons when you hardcopy a month window. Setting it to NIL suppresses this printing. Month windows are hardcopied at printer resolution in Koto, screen resolution in Lyric.

**CALHILITETODAY** [Variable]

Initially 'CIRCLE. This variable determines how today's date will be highlighted in a month window. The default is to draw a circle around it. If you set this to 'BOX, a light gray grid will be placed over the date. Setting this to NIL suppresses all date highlighting.

CALKEEPEXPIREDREMSFLG [Variable]

Initially NIL. If you set this to T, Calendar will not automatically delete reminders when they fire (they can still be deleted using the Delete menu command, above). The default action is to delete reminders when they fire, although they will remain visible until the window is redisplayed.

CALMONTHDEFAULTREGION [Variable]

Initially (32 32 868 700). This specifies the default position and size for month windows. If you set the size to a value small enough to allow several month windows side by side, the windows will tile left to right, bottom to top.

CALREMDISPLAYREGION [Variable]

Initially (200 400 300 400). This specifies the default position and size for reminder display windows.

CALTUNE [Variable]

When a reminder is activated, it will play the tune stored here (in PLAYTUNE format). This is initially a two-note "ding-dong". Set this to NIL if you want no audible warning. 1100's and 1132's have no hardware for sound.

CALUPDATEONSHRINKFLG [Variable]

Initially 'Never'. This means that Calendar will save your reminders on a file only when you explicitly click Update from a Day Browser. If set to 'Shrink, it will cause Calendar to save your reminder file automatically only when you shrink the Month window. This is useful when you are entering many reminders at the same time, but it means you must remember to explicitly shrink the month window or your reminders will be lost if your machine dies. If set to 'Always, causes Calendar to immediately save each reminder as soon as it is created.

You can also set these variables interactively by clicking on the box marked "Options" in any Month window. This brings up a freemenu similar to the TEdit expanded menu.

```

Calendar Options
Alert: Yes No
Keep expired rems.: Yes No
Auto. file update: Always Shrink Never
Alert delta: 0
Host & dir.: {DSK}<LISPPFILES>
Apply!

```

**Alert:** Specifies the default for the Alert field in the new reminder form. Sets the value of CALALERTFLG (described above).

**Keep expired rems.:** If set to No, the system will automatically delete reminders when they fire (although they remain listed in the month window until the next time you redisplay it). Sets the value of CALKEEPEXPIREDREMSFLG.

**Auto. file update:** Always means that the system will update the reminder file every time you create a new reminder. Shrink means update only when a month window is shrunken. Never means updates will be done only when you explicitly select Update from a Day browser. Sets the value of CALUPDATEONSHRINKFLG.

*Alert delta:* Sets the value of CALDEFAULTALERTDELTA.

*Host & dir.:* Sets the value of CALDEFAULTHOST&DIR.

After you have made the selections you want, click Apply! This sets the selections and closes the menu. If you don't want to make any changes, just close the menu (like closing any window). This preserves the previous settings even if you changed them in the menu. Any changes you make to these variables are not saved automatically in reminder files.

#### **IV. LIMITATIONS**

Day groups must begin and end in the same month.

The calendar algorithm is valid only for years between 1700 and 2100.

#### **V. KNOWN BUGS**

Today-circling function occasionally fails to erase the old day.

#### **VI. FUTURE PLANS**

Automatic scheduling.

Automatic communication with other Calendars.



---

---

**CANVASCONVERTER**

---

---

By: Stephen Knowles (Stephen Knowles:49/89/636/13:Siemens AG)

Partly based on work by:

Matthias Schneider-Hufschmidt (Matthias Schneider-Hufschmidt:ZTISOF:SIEMENS)

Giselbert Schramm (Giselbert Schramm:ZTISOF:SIEMENS)

Uses: BITMAPFNS

This document last edited on 19-Sep-88 13:32:21.

## INTRODUCTION

This module enables the transfer of bitmaps between the Envos Lisp and Xerox ViewPoint environments. The medium used for the transfer is an NS file server (i.e. a file drawer which can be accessed by both environments). The possibility of transferring Lisp bitmaps into the ViewPoint environment is particularly useful for documenting Lisp applications.

## MODULE EXPLANATIONS

There are essentially two major functions:

(IL:WRITECANVAS *BITMAP FILE*) [Function]

This function writes the *BITMAP* on to *FILE* and makes *FILE* of type ViewPoint Canvas, whereby *FILE* must be on an NS file server.

(IL:FETCHCANVAS *FILE*) [Function]

This function reads *FILE* into a Lisp bitmap, whereby *FILE* must be on an NS file server.

Additionally there are two auxiliary functions to aid in the use of the above two functions.

(IL:SNAPBM) [Function]

and

(IL:CANVAS-FROM-WINDOW *WINDOW FILE*) [Function] -

## EXAMPLES

All examples must be typed into an INTERLISP exec.

To write a canvas of a Lisp screen region:

(WRITECANVAS (SNAPBM)

```
'{NSfileServer:Domain:Organization}<FileDrawer>Folder>TESTFILE)
```

To write a canvas of a Lisp window:

```
(CANVAS-FROM-WINDOW (WHICHW)
```

```
'{NSfileServer:Domain:Organization}<FileDrawer>Folder>TESTFILE)
```

To read a canvas into a Lisp bitmap:

```
-(SETQ X (OPENSTREAM
```

```
'{NSfileServer:Domain:Organization}<FileDrawer>Folder>TESTCANVAS 'INPUT))
```

```
(EDITBM (SETQ LISPBITMAP (FETCHCANVAS X)))
```

```
(CLOSEF X)
```

#### CAVEAT

When fetching a canvas, there is a 50-50 chance that the Lisp bitmap will be O.K. It could, however, come out distorted (this is due to the differing ways in which ViewPoint and Lisp handle bitmaps, Lisp uses 16 complement, ViewPoint 32 complement - or something like that). If this should be the case, simply increase the canvas width in ViewPoint by 5 millimeters (approx. 16 pixels) and repeat the fetching process.

Unfortunately in the Lyric version if one repeatedly wrote a canvas with the same name, the file server somehow got mixed up and set the file-info of the folder above the canvas into "type = canvas"! One could put this right with the (SETFILEINFO...) function in Lisp, although under normal circumstances one does not write out a canvas repeatedly with the same name any way. I have been unable to test the behaviour in MEDLEY.

Compatibility has only been tested up to ViewPoint 1.1.

---



---

**CD**


---



---

By: Henry Thompson (HThompson.pa@Xerox.com)

This document last edited July 6, 1988.

## INTRODUCTION

The file CD implements a UNIX\*-style facility for manipulating the connected directory. It also insures that the connected directory is always displayed.

### CD PATTERN

[Exec command]

## MODULE EXPLANATIONS

CD is defined as a command which allows low-overhead means of effecting many common changes of connected directory. Its behaviour is partly conditioned by three global variables:

CD.DEFAULT.HOST [Variable]

CD.DEFAULT.PREFIX [Variable]

CD.DEFAULT.USER [Variable]

CD.DEFAULT.HOST defaults to DSK. CD.DEFAULT.PREFIX defaults to the name (e.g. DSK) of the local disk volume on a Dandelion, otherwise NIL. CD.DEFAULT.USER defaults to the value of USERNAME, and is updated automatically after GREETing.

The value of CD is always a CONS-pair of the old and new connected directories.

On hosts which support some form of sub-directory, CD needs to know the character which is used to separate sub-directories. The table CD.OS.SEPRS is an a-list which determines this mapping - it is initialised to map UNIX\* and VMS to "/" and DSK, NS and IFS to ">". To enter this table it looks up the host first in CD.OS.SEPRS directly, then via NETWORKOSTYPES. In the documentation which follows, ">" means whatever the separator is for the relevant host.

The possibilities for *pattern* are as follows:

### *empty*

Connects to the directory determined by the conjunction of CD.DEFAULT.HOST, CD.DEFAULT.PREFIX and CD.DEFAULT.USER.

### *{anything}*

Interprets *pattern* as a complete directory specification, and connects to it.

### *<anything*

Interprets *pattern* as a directory specification to be qualified by CD.DEFAULT.HOST and CD.DEFAULT.PREFIX, and connects to it. For example if CD.DEFAULT.HOST is {server} and CD.DEFAULT.PREFIX is NIL, then CD <dir>sdir> is equivalent to CD {server}<dir>sdir>, whereas

if CD.DEFAULT.PREFIX was /user and server was known to be running UNIX\*, then CD <dir/sdir> would be equivalent to CD {server}</user/dir/sdir>.

.>rest

Equivalent to CD rest. This is purely for compatability with UNIX\*.

..>rest

-Equivalent to peeling off one (sub-)directory from the currently connected directory, followed by CD rest. For example, if connected to {server}<dir>sdir>, then CD ..>sdir1 is equivalent to CD {server}<dir>sdir1>. Note that because of common lisp reader peculiarities, you cannot use .. alone under a common lisp read-table. The synonym << can be used instead.

*otherwise*

Treat *pattern* as a further specialisation of the current directory, and connect to the resulting sub-directory. For example, if connected to {server}<dir>sdir>, then CD ssdir is equivalent to CD{server}<dir>sdir>ssdir>.

Note that throughout, the closing ">" is optional.

### Menu Interface

At any time you can left button in the window displaying the current connected directory, and see a menu of all the directories you have yet been connected to. Selecting one will move you there. You can also shift-select out of this menu into the current input stream. This latter is very useful when typing file names.

Middle buttonning in the directory display window will give you a menu of directories, followed by a menu of Connect/Browse/Delete. Connect does so, Browse brings up a file browser and Delete removes the directory from subsequent menus.

---

\*UNIX is a trademark of Bell Laboratories.

---

---

## CHATEMACS

---

---

By: Randy Gobbel (Gobbel.pa)

requires:CHAT, chatemacs.elc (GnuEmacs Lisp program)

This document last edited on August 24, 1987

### INTRODUCTION

ChatEmacs, in conjunction with the `chatemacs.elc` module for GnuEmacs, enables use of the mouse for scrolling and selection in GnuEmacs. It also allows use of the META key for escape-prefix commands and automatically switches Chat in and out of Emacs mode when entering and leaving the editor.

### DETAILS

After loading ChatEmacs, typing META-char will send an ESCAPE character, followed by the vanilla character. CTRL-META-char sends an ESCAPE followed by CTRL-char. Once ChatEmacs is active, most Emacs commands should require only one keystroke. Since Emacs was originally designed for terminals with a META shift key, this makes the Emacs command set somewhat more regular and easier to remember. For example, scrolling forward and backward will be on CTRL-V and META-V, respectively.

In order to enable mouse actions, first load CHATEMACS.LCOM into Interlisp. After opening a Chat connection and running GnuEmacs, either load `chatemacs.elc` manually (by giving the ↑Xload command), or add the following line to your GnuEmacs init file (`.emacs`):

```
(load "chatemacs")
```

After loading `chatemacs.elc`, the title bar on your Chat window should say "Emacs ON". If not, middle-buttoning the "Emacs" menu item in your Chat window will enable mouse events to be sent to Emacs. After ChatEmacs has been activated for the first time, the Chat window's title bar will always indicate whether Emacs mode is on or off. If your mouse clicks don't seem to be taking effect, check the title bar first!

Automatic switching frees the user from having to manually turn ChatEmacs on and off when using Emacs. In most circumstances (see exceptions below) automatic switching will not interfere with other Chat operations, and can be left enabled. Auto-switching is controlled by:

CHATEMACS.SWITCH.ENABLED [Variable]

When this variable is non-NIL, Chat will respond to a sequence of two consecutive ESCAPEs by toggling the flag that controls mouse event sending. The state of the flag is noted in the window's title bar, just as if the menu command had been executed. CHATEMACS.SWITCH.ENABLED is defaulted to NIL.

auto-switch-enabled

[GnuEmacs Variable]

This variable controls auto-switching on the GnuEmacs side of the Chat connection. If it is non-nil, GnuEmacs will send a switch command when chatemacs.elc is loaded, and another when exited via a  $\uparrow X$ - $\uparrow C$  command.

### Using Emacs with the mouse

The chatemacs.elc module, at the GnuEmacs end of the connection, determines the interpretation of mouse clicks. The current user interface is more complicated than I would like, and suggestions for improvements are welcome.

The most basic operations are fairly simple: left button in a text buffer moves the Emacs "point" to wherever the cursor is pointing. Right button moves the mark (the typein cursor will move for a couple of seconds just to show you where you've just put the mark), and copies the new region to the kill buffer (for use with "shift-select," see below).

Scrolling with the mouse works more or less as in Interlisp, with the scrollbar being the right-hand part of the screen past column 80. Alternatively, holding down the META key makes the entire text area act as a "scrollbar". As in most Envos environments, left button scrolls the line that the mouse is pointing to to the top of the window, right button moves the top line down to the mouse cursor, and middle button "thumbs", taking the vertical displacement of the mouse cursor as an offset into the file (i.e., top line = beginning of file, bottom line = end of file).

Shift- and control- mouse clicks perform editing operations: shift-left copies the contents of the kill buffer to wherever the mouse is pointing (the closest thing to Interlisp shift-select I could come up with). Control-left and control-right kill from point to where the mouse is pointing (sort of like control-select). Control-shift-left moves the mark without copying anything to the kill buffer.

Mouse clicks in the mode line and minibuffer do things that were inherited from il-mouse's ancestor, a package for the BBN Bitgraph terminal. Maybe you will find them useful. They are: The modeline acts like a sideways scrollbar, left = top. In the minibuffer, left button is equivalent to typing META-X, middle button evals an expression you type in, and (beware!) right button suspends Emacs (equivalent to typing  $\uparrow X \uparrow Z$ ).

As mentioned above, the current user interface is sort of, how shall I say, "gnarly." If you have better ideas, please let me (Gobbel.pa) know.

---

---

**CHATSERVER**

---

---

By: Larry Masinter (Masinter.PA@Xerox.COM)

This document last edited on September 7, 1988.

**REQUIREMENTS**

CHATSERVER-NS Requires: CHATSERVER and COURIERSERVE.

CHATSERVER-RS232 requires: CHATSERVER and (DLTTY or DLRS232C). As of this date, CHATSERVER-RS232 hadn't been tested with Medley.

CHATSERVER-TCP requires: CHATSERVER and TCP. As of this date, CHATSERVER-TCP is unreliable: the chat server sometimes leaves open the connection and will not open another one.

In general, a protocol chatserver requires CHATSERVER and a protocol converter. Sources for TCP server available.

CHATSERVER also loads LispUsers modules CL-TTYEDIT and SIMPLECHAT.

The module PREEMPTIVE is useful in conjunction with CHATSERVER but not required.

**INTRODUCTION**

CHATSERVER is a general facility that allows a Lisp workstation to be controlled from a dumb terminal. In addition to CHATSERVER, you will need a protocol driver: something that connects the CHATSERVER to a communication protocol. The various protocol drivers are the mechanism by which CHATSERVER can be controlled; versions include using XNS via CHATSERVER-NS, TCP/IP TELNET protocol via CHATSERVER-TCP, and RS232 via CHATSERVER-RS232. CHATSERVER-NS is the most reliable, although CHATSERVER-RS232 has worked reliably in the Lyric release.

The server implements password protection using the same mechanism as IDLE. There is another variable, CHATSERVER.PROFILE, which gets searched first for ALLOWED.LOGINS so that you can have a different setting.

IL:CHATSERVER.PROFILE [Variable]

The value of the variable CHATSERVER.PROFILE appended to the front of IDLE.PROFILE when determining login options etc for the chatserver. The property IDLE.ONLY is also consulted; if T, chatserver only allows connections when machine is in idle mode.

**Example:**

```
(SETQ CHATSERVER.PROFILE '(ALLOWED.LOGIN (T) IDLE.ONLY T))
```

means to allow only the previously logged in user, and then, only when in IDLE mode.

QUIT [Exec Command]

The QUIT exec command exits a chatserver session. It signals an error if you are not running in a chatserver session.

**Documentation for CHATSERVER-NS:**

CHATSERVER-NS implements the Xerox Network Systems GAPTNET protocol. It allows connection to a machine running Medley from other machines that implement this protocol, including Viewpoint (using ViewpointChat), External Communication Service servers (which allow dial-in from remote terminals into an XNS network), XDE workstations and other Interlisp-D implementations (Koto, Lyric, Medley.)

Gaptelnet is a courier server program, and so requires the COURIERSERVE lispusers module (which it loads automatically). The following function is part of COURIERSERVE

(COURIER.START.SERVER) [Function]

This "starts" the courier server process which listens for connection attempts. It is necessary to call this (once) before you can CHAT to your Lisp workstation. (If the process dies for some reason, you will not be able to CHAT until you restart the process.)

**Documentation for CHATSERVER-TCP:**

This module was an attempt to implement a TCP/TELNET server. Unfortunately, the mechanism by which it waits for a connection is buggy, and it does not negotiate terminal characteristics properly with the client calling workstation. It is therefore unreliable & may need to be restarted. Using telnet from a Sun to Lisp I've found it was necessary to explicitly tell the Sun not to echo, to send character at a time, etc.

(TCPCHATSERVER) [Function]

It is necessary to call this function to spawn the process that waits for TCP connections.

**Documentation for CHATSERVER-RS232:**

The CHATSERVER-RS232 module attempts to allow for connections on the RS232 or TTY port on an 1186 or 1108. It uses DLRS232.

(RS232CHATSERVER) [Function]

Spawns a process waiting for a character to be typed on the RS232 port, and then starts a chatserver session.

**Other notes:**

The server runs a standard (XCL) exec. Note that you can't do graphics; the debugger will not attempt to open a window, only the type-in commands are available, ED will give you the "teletype" editor. Interrupt characters enabled are ↑E, ↑D, DEL, ↑B, ↑H and ↑T. (Note that currently interrupts are only processed when they are read, and there is no way to interrupt a run-away process.)

Typeout uses a "---more---" style: after (PAGEHEIGHT) lines, the system will prompt you with a "---more---". Type any character, and the more will be erased.

Chatserver assumes you are chatting from a DM2500 emulator, and treats font changes as a switch between bold and regular as appropriate.

The PREEMPTIVE Lispusers module is useful when running chatserver, because it will keep the running process from blocking out the typein process. For some protocol drivers (and the NS server in particular), this is necessary to avoid timeouts.



CHATSERVER advises various facilities in the environment that normally create menus to check to see if the "controlling" keyboard is not the workstation console; these facilities include TTYIN, the editor, the debugger, CHAT. Thus, calls to the editor use the teletype-style editor from Interlisp, while FIX does not generally allow character editing.

---

---

**CHECKPOINT**

---

---

By: Herb Jellinek (Jellinek.pa@Xerox.com)

This document was last edited on January 8, 1988.

**INTRODUCTION**

CHECKPOINT provides a Cedar-like checkpoint facility to the Envos Lisp system. Performing a checkpoint freezes the state of virtual memory; if the machine subsequently crashes, or the user logs out and restarts Lisp again, work will progress from the point at which the checkpoint was made.

This is different from the effect of SAVEVM in that CHECKPOINT lets you freeze the state of your system and guarantee that it will remain consistent until you decide otherwise. The effect lasts across (LOGOUT T), rebooting, and so on.

CHECKPOINT is based on calls to IL:VMEM.PURE.STATE and IL:SAVEVM, but features a convenient user interface.

**USING IT**

If you have WHO-LINE loaded, CHECKPOINT will install a "Ckpt" item displaying time of last checkpoint or "OFF" if no checkpoint is in effect. Clicking this item will pop up a menu asking you whether to make a new checkpoint or disable the current one, if any. Turning off checkpointing means that the virtual memory file will proceed to be written once again.

ckpt on/off?

[Exec Command]

If the argument on/off? is :ON or NIL, a checkpoint is taken, if :OFF, the checkpoint is disabled, and if :STAT, the checkpoint status is printed.

---

---

**CL-TTYEDIT**

---

---

By: Larry Masinter (Masinter.PA@Xerox.COM)

This document last edited on November 24, 1987.

**INTRODUCTION**

This file patches the TTY editor so that it is a little more usable in Lyric/Medley for non-Interlisp sources. In particular, it changes the TTY editor so that EDITRDTBL is no longer used; the read table in effect at the time the ttyeditor is invoked is used instead (\*READTABLE\*).

It patches the main editor loop (EDITCOM) so that the package and case of edit commands are ignored, i.e., if you type in the P command, it doesn't care whether it is XCL-USER::P or IL:P or |p|.

It patches EDITFPAT (which takes "find" patterns) so that you can specify patterns with --, &, = =, \*ANY\* in any package, and use --- instead of .. (since symbols consisting entirely of dots are not allowed in CL readtables.)

This file is especially useful if you are talking to another machine using CHATSERVER and need to edit something on the remote machine; since the CHAT connection is character only, you can't run (and the system doesn't attempt to run) SEDIT.

---



---

## COMPARESOURCES

---



---

By: Bill van Melle (vanMelle.pa@Xerox.com)

### INTRODUCTION

COMPARESOURCES is a program for comparing two versions of a Lisp source file for differences. The comparison is completely brute-force: COMPARESOURCE reads the complete contents of both files, and compares all the expressions for differences. The files need not be ones produced by MAKEFILE, as COMPARESOURCE reads the contents with READFILE; however, the program is tuned for files of the type produced by MAKEFILE.

### HOW TO USE IT

The interface consists of a single function:

(COMPARESOURCES *FILEX FILEY EXAMINE DW? LISTSTREAM*) [Function]

Compares the files named *FILEX* and *FILEY* for differences. For each type of file object (function, variable, record, etc), COMPARESOURCE identifies which objects of that type differ, and for each such object prints on *LISTSTREAM* a comparison using the function COMPARELISTS. If an object exists on only one of the two files, this fact is noted instead by the message "*name* is not on *file*".

If *DW?* is true, COMPARESOURCE calls DWIMIFY on each function body before performing the comparison. This is useful for comparing a file made with CLISP prettyprinted with one made without.

If *EXAMINE* is true, COMPARESOURCE calls the editor to allow you to more closely examine expressions that differ. Its value is either T, meaning call the editor in all cases, or an atom or a list of atoms chosen from among the following:

OLD	Call the editor for changed objects that are on both files.
NEW	Call the editor for objects that are on only one file.
MISC	Call the editor for changed but otherwise unclassified expressions.

In the OLD and MISC cases, the editor is called on a list of two elements, the two expressions. In the NEW case, the editor is called on just the single new expression.

The value returned by COMPARESOURCE is a list whose elements are of the form (type . names), listing the names by type of all objects found to be different. Expressions of no particular type are identified collectively as "(Other --)".

### FORM OF THE OUTPUT

The output of COMPARESOURCE is in several sections. First, all functions are compared. Then expressions of other types (variables, macros, etc) are compared. When a difference is found, COMPARESOURCE prints the name of the object and calls COMPARELISTS, the same Interlisp function called by COMPARE and COMPAREDEFS. Finally, expressions inside of DECLARE: forms are recursively analyzed in a separate section in the same fashion. All DECLARE: forms of the same applicability (e.g., EVAL@COMPILE DONTCOPY) are handled in the same subsection.

The output of COMPARELISTS takes one of three forms. The usual form is an abbreviated printing of the two expressions with equal elements in the two structures denoted by "&" or "-n-" for a subsequence of *n* identical expressions. Identical elements are printed only for purposes of establishing the context of differences. For example,

COMPARESOURCES:

```
(LAMBDA -3- (PROG -16- (for -10- (COND (& (printout & T -4-) -2-)))
(TERPRI --) &))
(LAMBDA -3- (PROG -16- (for -10- (COND (& (printout & -4-) -2-)))
&))
```

indicates that in the function COMPARESOURCES, an extra argument was added to a printout form, and a (TERPRI --) expression was added before the final element of the PROG. The first 17 elements of the PROG form were unchanged, as were the first 11 and last 2 of the for.

A more abbreviated form of output occurs when the expressions differ only in a global substitution. In this case, COMPARELISTS prints "(x -> y)" to denote that all occurrences of *x* in the first expression were replaced by *y* in the second expression, and there were no other changes.

Finally, COMPARELISTS prints "SAME" if the expressions are "the same". Since COMPARESOURCES only calls COMPARELISTS when the two expressions are not EQUAL, the output SAME specifically means that the expressions differ only in the bodies of comments (which COMPARELISTS ignores).

#### USER EXTENSIONS

COMPARESOURCES already "knows" about several kinds of file package objects, including FNS, VARS, MACROS, RECORDS, and PROPS. Any expression not identifiable as some particular type is compared as a vanilla expression. You can extend the set of types it knows about by adding to the following list:

COMPARESOURCETYPES [Variable]

The elements of this list are lists of the form

```
(TYPE PREDICATEFN COMPAREFN IDFN DESCRIPTION)
```

as follows:

- |             |  |
|-------------|--|
| TYPE        | The file package type of the object (or whatever name you wish to give it in the case of fictitious object types).   |
| PREDICATEFN | A function of one argument, a single top-level expression as read from the file, that returns true if the expression is of the desired type.   |
| COMPAREFN   | A function of three arguments, one expression from each file (both guaranteed to have satisfied the PREDICATEFN), and the listing stream. COMPAREFN should compare the two expressions in some appropriate way, printing its results to the listing stream. A typical COMPAREFN calls the function COMPARELISTS on some subform of the expressions. If COMPAREFN is NIL, COMPARELISTS is used. |
| IDFN        | A function of one argument, an expression, that returns the "name" of the object described by the expression. Two expressions are assumed to define the same object if their names are EQUAL. The name corresponds roughly to  |

a file package name. For example, for type VARS it is the variable name; for type PROPS it is a pair (atom propname). If IDFN is NIL, CADR is used.

**DESCRIPTION** A string identifying the kind of object, for use in the comparison printout. If DESCRIPTION is NIL, (L-CASE TYPE T) is used.

---



---

## COMPARETEXT

---



---

By Mike Sannella. Tested in Medley by Larry Masinter (Masinter.PA@Xerox.COM)

Uses TEDIT.LCOM, GRAPHER.LCOM

### INTRODUCTION

COMPARETEXT is a rather non-standard text file comparison program which tries to address two problems: (1) the problem of detecting certain types of changes, such as detecting when a paragraph is moved to a different part of a document; and (2) the problem of showing the user what changes have been made in a document.

The text comparison algorithm is an adaptation of the one described in the article "A Technique for Isolating Differences Between Files" by Paul Heckel, in CACM, V21, #4, April 1978. The main idea is to break each of the two text files into "chunks" (words, lines, paragraphs, ...), hash each chunk into a hash value, and match up chunks with the same hash value in the two files. This method detects switching two chunks, or moving a chunk anywhere else in the document.

### COMPARING TEXT FILES

Two text files can be compared with the following function:

(COMPARETEXT *NEWFILENAME* *OLDFILENAME* *HASH.TYPE* *GRAPH.REGION*) [Function]

*NEWFILENAME* and *OLDFILENAME* are the names of the two files to compare. The order is not important, except that in the resulting graph the *NEWFILENAME* information will appear on the left, and the *OLDFILENAME* info on the right.

*HASH.TYPE* determines how "chunks" of text are defined; how fine-grained the comparison will be. This can be *PARA* to hash by paragraphs (delimited by two consecutive CRs), *LINE* to hash by lines (delimited by one CR), or *WORD* to hash words (delimited by any white space). *HASH.TYPE* = *NIL* defaults to *PARA*.

*GRAPH.REGION* is the region on the display screen used for the file comparison graph. If *GRAPH.REGION* = *NIL*, the system asks the user to specify a region. If *GRAPH.REGION* = *T*, a region in the lower left corner is used.

COMPARETEXT creates a graph with two columns. Each column contains the file name of one of the files, and lists the chunks from that file. Each chunk is represented by an atom *NNN:MMM*, where *NNN* is the file pointer of the beginning of the chunk within the file, and *MMM* is the length of the chunk. Lines are drawn from one column to the other to show which chunks in one file are the same as those in the other file. Chunks with no lines going to them do not exist in the other file. [Note: a series of chunks in one file which are the same as a series of chunks in the other file are merged into one big chunk. A series of unconnected chunks is also merged.]

Pressing the *LEFT* mouse button over one of the chunk nodes causes the node to be boxed, and a Tedit window to be opened on the file, with the appropriate text selected. If a Tedit window to the file is already active, the selection is simply moved.

Pressing the MIDDLE mouse button over a chunk node raises a pop-up menu with the items: PARA, LINE, and WORD. If one of these is selected, COMPARETEXT is called to compare the selected chunk with the last selected chunk (the one that is boxed), using the hash type selected, and create a new graph window.

If the mouse is buttoned outside of the PARA/LINE/WORD menu, no comparison is done, but the selected node is boxed. The PARA/LINE/WORD menu is always brought up a little away from the cursor, so pressing double-MIDDLE-button over a chunk node is a way to change the boxed node without calling Tedit.

Important note: white space (space, tab, CR, LF) is used to delimit chunks, but is ignored when computing the hash value of a chunk. Therefore, if two paragraphs are identical except that one has a few extra CRs after it, they will be considered identical by COMPARETEXT.



---

---

**COMPILEBANG**

---

---

By: Nick Briggs (Briggs.pa@Xerox.com)

Required by TRILLIUM

This provides an interface to the compiler that avoids the interview for the common cases of in-core compilation. It contains a single function `COMPILE!`, and the Lisp<sub>x</sub> and edit macros `C`:

`(COMPILE! X NOSAVE NOREDEFINE PRINTLAP)` [Function]

Calls the compiler to compile `X`. If `X` is a litatom, its definition is compiled and stored in the function cell unless `NOREDEFINE`, and the old definition if any is saved on the property list unless `NOSAVE`. No printing of lap or machine code is done unless `PRINTLAP`.

Thus, to simply compile the function `BAR`, do `COMPILE!(BAR)`.

`X` may also be a list form. In this case, `COMPILE!` assumes that the user is interested just in seeing how that form would compile. The form is embedded in a Lambda expression and compiled. Of course, there is no function-cell to be stored into or saved.

`C` [Lisp<sub>x</sub> Macro]

The LISP<sub>x</sub>MACRO `C` calls `COMPILE!`, with `PRINTLAP` on, on the next element of the input line. Thus, `C BAR` will compile, redefine, and save the old definition for `BAR`.

`C (CONS)` will show how a call to `CONS` would compile.

The editmacro `C` calls `COMPILE!` on the current expression if it is a list, or on the form of which the current expression is an element.

---

---

**COURIERDEFS**

---

---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

\_COURIERDEFS contains a procedure-less Courier program, called INTERLISP, which defines several Envos Lisp types as Courier constructed types or as new Courier primitive types (via a COURIERDEF property) for use with Courier server and client programs. The defined Envos Lisp types include:

<b>ATOM</b>	Converts to a string on writing and converts to an atom on reading.
<b>FONT</b>	Converts a FONTDESCRIPTOR to a record describing the font on writing and convert the record back to a FONTDESCRIPTOR on reading.
<b>REGION</b>	A sequence of INTEGER.
<b>POSITION</b>	Converts a POSITION record to two integers on writing and converts back to a POSITION record on reading.
<b>NUMBER</b>	Like INTEGER but can also be NIL.
<b>BRUSH</b>	Converts the various possibilities for a brush (NIL, INTEGER, BRUSH RECORD etc.) to a CHOICE record on writing, converts back to original specification on reading.
<b>OPERATION</b>	An ENUMERATION of NIL, REPLACE, PAINT, INVERT or ERASE.
<b>TEXTURE</b>	Converts a TEXTURE, NIL or T to a CARDINAL on writing, returns a CARDINAL on reading.

This file is loaded by several other modules that define Courier servers and clients. A Courier program can use the types defined in the INTERLISP program by using the INHERITS slot in the Courier program definition.

---

---

**COURIEREVALSERVE**

---

---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: COURIERSERVE

COURIEREVALSERVE implements both the client and server routines for the simple remote evaluation server described in the COURIERSERVE documentation.

The module defines two user functions:

(REMOTEEVAL *FORM* *COURIERSTREAM* [*NOERRORFLG*]) [Function]

(REMOTEAAPPLY *FN* *ARGS* *COURIERSTREAM* [*NOERRORFLG*]) [Function]

*COURIERSTREAM* is obtained by calling *COURIER.OPEN* to connect with a host that is running the Courier server and has *COURIEREVALSERVE* loaded. If the *NOERRORFLG* is non-NIL, it is returned if an error is signaled by the remote host, otherwise the functions generate an error.

Due to the removal of *ERRORN* as of the Lyric release, the error handling is not as informative as in earlier versions. If you are connected to a pre-Lyric host, errors will work as before, otherwise instead of signaling the actual remote error (eg. "*Undefined car of form*") the generic "*Remote evaluation error!*" error is raised. This is to maintain backward compatibility in the *EVAL* Courier program. Hopefully, this will be replaced by a new version of the *EVAL* program designed to correctly remote the new condition-based error handler.

---



---

## COURIERIMAGESTREAM

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: COURIERSERVE, COURIERDEFS and BITMAPFNS

COURIERIMAGESTREAM implements a Courier client and server program which allows remote hosts to do *image stream* manipulations on other workstations via the network. To do this, it defines the COURIER virtual image stream type which allows the user to manipulate remote image streams through local image streams.

### THE IMAGESTREAM COURIER PROGRAM

The module defines a Courier program called IMAGESTREAM (which inherits from the INTERLISP Courier program defined in COURIERDEFS). For each IMAGEOP in the IMAGEOPs definition, there is an equivalent Courier procedure in the IMAGESTREAM program. The module contains the code for both the Courier client and server.

### OPENING AND CLOSING REMOTE IMAGE STREAMS

Remote image streams can be opened using either the COURIER image stream type or using direct Courier calls.

#### The COURIER Image Stream Interface

Remote Courier image streams can be opened using:

```
(SETQ COURIERSTREAM (COURIER.OPEN HOST)
 (OPENIMAGESTREAM COURIERSTREAM 'COURIER OPTIONS))
```

which returns an image stream. The OPTIONS can include FILE and IMAGETYPE which are passed to OPENIMAGESTREAM on the remote host and if not supplied, a nameless DISPLAY image stream is opened. All other options are passed to the remote image stream. The image stream can be closed using CLOSEF.

#### The Courier Procedure Call Interface

Courier image streams can also be opened using Courier procedure calls from any Courier client with the Courier procedure:

```
(OPEN 0 (FILE IMAGETYPE) RETURNS (HANDLE) REPORTS NIL)
```

which is invoked from Lisp by doing:

```
(COURIER.CALL COURIERSTREAM 'IMAGESTREAM 'OPEN FILE IMAGETYPE OPTIONS)
```

where FILE, IMAGETYPE and OPTIONS are similar to the arguments to OPENIMAGESTREAM.

This call will return a handle to be used with the remainder of the IMAGESTREAM procedures. To close an image stream from a Courier client use the Courier procedure:

(CLOSE 1 (HANDLE) RETURNS NIL REPORTS NIL)

which is invoked from Lisp by doing:

(COURIER.CALL COURIERSTREAM 'IMAGESTREAM 'CLOSE HANDLE)

#### **DIFFERENCES BETWEEN IMAGEOPS AND IMAGESTREAM COURIER PROCEDURES**

All of the IMAGEOPs are implemented in the COURIER image stream type as it merely passes the call to the IMAGEOPs of another image stream type on the remote host. No error checking is done, so invoking an illegal IMAGEOP will cause a Courier rejection of the call.

The arguments to the IMAGESTREAM Courier procedures are generally in the same order as the arguments to the various IM\* functions which implement an image stream (stream argument first). An exception is BITBLT (and SCALED BITBLT) which is defined as follows:

(BITBLT 32 (HANDLE BULK.DATA.SOURCE LEFT BOTTOM WIDTH HEIGHT  
SOURCETYPE OPERATION TEXTURE CLIPPINGREGION)

The BULK.DATA.SOURCE argument is used to transfer the bitmap using WRITEBINARYBITMAP. This is only relevant to direct Courier calls, the COURIER image stream BITBLT operation hides the differences.

When using the COURIER image stream type, the STRINGWIDTH, CHARWIDTH etc. IMAGEOPs are handled locally, not via Courier calls, to improve efficiency.

#### **IMAGESTREAM PROGRAM VERSIONS**

The current implementation of the IMAGESTREAM Courier program is version 1. This module also has the previous version of the program (0) defined as OLDIMAGESTREAM (just the procedure definitions that differ, it inherits from IMAGESTREAM). This allows the current version of the program to accept calls from older versions, but not vice-versa. However, the new version of the IMAGESTREAM Courier program can be loaded and used with the old (pre-Lyric) functions.

---



---

## COURIERSERVE

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

COURIERSERVE implements a Courier server process for Envos Lisp allowing other hosts to make Courier calls into the workstation. The server supports both multiple Courier stream connections as well as *expedited* (single packet) and *broadcast* calls.

### STARTING A COURIER SERVER

The Courier server can be started by evaluating:

(COURIER.START.SERVER [RESTART]) [Function]

Once the server is running, it can be invoked by a remote host client using COURIER.OPEN for a Courier stream connection or by using COURIER.EXPEDITED.CALL or COURIER.BROADCAST.CALL for expedited calls. The functions for making Courier client calls from Lisp are documented in the *Interlisp-D Reference Manual* (pages 31.15–31.26).

(COURIER.RESET.SOCKET) [Function]

(Re)Opens and closes the Courier socket. Not normally a user routine, this function is called by COURIER.START.SERVER but it can be called directly if "*socket already open!*" errors persist on the Courier socket (5).

### DEFINING A COURIER SERVER FUNCTION

Defining a Courier server program is identical to defining a client program except for the additional field IMPLEMENTEDBY in each procedure in the PROCEDURES section of the Courier program definition:

#### PROCEDURES

```
((LAYOUT 0 (GRAPHNODES ROOTIDS FORMAT FONT MOTHERD PERSONALD FAMILYD)
  RETURNS (GRAPH)
  REPORTS (LAYOUT.ERROR)
  IMPLEMENTEDBY GRAPH.REMOTELAYOUT))
```

The order of the RETURNS, REPORTS and IMPLEMENTEDBY fields is significant and should be maintained.

The *server function*, named in the IMPLEMENTEDBY field, is invoked when a Courier call to the procedure is made. The server function is applied to the Courier stream, the Courier program and the Courier procedure followed by the arguments named in the Courier definition. The arguments for GRAPH.REMOTELAYOUT would be (COURIERSTREAM PROGRAM PROCEDURE GRAPHNODES ROOTIDS FORMAT ...).

Note that the COURIERSTREAM, PROGRAM and PROCEDURE arguments are not necessarily used, they are made available for implementing special servers.

## RETURNING VALUES FROM A COURIER PROGRAM

Results or errors can be returned by a Courier server function by one of two different methods. In the usual, simple case, the server function can return as its result a list starting with one of RETURN, ABORT or REJECT followed by the appropriate values.

For the RETURN result, the tail of the list should be the results as defined in the Courier procedure definition, eg. (RETURN 23 "John").

For the ABORT result, the tail of the list should contain the reason for the abnormal termination (as defined in the Courier program), followed by any error arguments, eg. (ABORT NAME .NOT .FOUND "John").

For the REJECT result, the tail of the list should contain the rejection error as defined in the Courier standard. The only rejection that should occur inside a server function should be UNSPECIFIED if the program needs to reject for any reason. The other rejection types are handled by the Courier server.

Alternatively, the server function can return results directly to the Courier stream and return NIL as its result. To return results directly to the Courier stream use:

(COURIER.RETURN <i>COURIERSTREAM PROGRAM PROCEDURE RESULTLST</i> )	[Function]
(COURIER.ABORT <i>COURIERSTREAM PROGRAM ERROR RESULTLST</i> )	[Function]
(COURIER.REJECT <i>COURIERSTREAM ERROR RESULTLST</i> )	[Function]

## EXPEDITED AND BROADCAST COURIER CALLS

The Courier server allows expedited and broadcast Courier calls. The only difference the server function would see if invoked due to an expedited call is that the Courier stream it is handed is actually a record containing an XIP packet and a socket. If the server function does not use the Courier stream directly, then this difference is invisible.

If the server function actually needs a Courier stream to operate (eg. an NS CHAT server), then it should probably include an USE.COURIER abort error in its definition. If the server function needs a Courier stream due to *bulk data* arguments, this will be trapped in the Courier server itself, which will reject appropriately and not invoke the server function.

## USING BULK DATA IN A SERVER FUNCTION

If a server function takes a bulk data argument (either BULK.DATA.SINK or BULK.DATA.SOURCE), it is handed an open bulk data stream for that argument when invoked. If the server function returns a result by returning one of the RETURN or ABORT forms as its result, the bulk data stream will be closed automatically. If the server function returns results directly to the Courier stream using COURIER.RETURN or COURIER.ABORT, then the server function must first close the bulk data stream using:

(CLOSE.BULK.DATA <i>STREAM [ABORTFLG]</i> )	[Function]
---	------------

The CLOSEF function does not work on the bulk data stream argument and using it will hang the Courier connection. Only the *immediate* bulk data transfer type is handled. NULL, ACTIVE or PASSIVE bulk data transfer types will cause a Courier rejection of type UNSPECIFIED.

**SIMPLE SERVER DEFINITION**

Below is the Courier definition for a simple evaluation server. The two functions EVAL.REMOTE and APPLY.REMOTE are all that would need to be defined to make the server run:

```
((1105 0)
  TYPES      ((SEXPR STRING)
              (FN STRING)
              (ARGS (SEQUENCE SEXPR))
              (ERRORN (RECORD (ERROR.NUMBER CARDINAL)
                               (ERROR.MESSAGE SEXPR))))

  PROCEDURES ((EVAL 0 (SEXPR)
                  RETURNS (SEXPR)
                  REPORTS (REMOTE.EVAL.ERROR REMOTE.READ.ERROR)
                  IMPLEMENTEDBY EVAL.REMOTE)
              (APPLY 1 (FN ARGS)
                  RETURNS (SEXPR)
                  REPORTS (REMOTE.APPLY.ERROR REMOTE.READ.ERROR)
                  IMPLEMENTEDBY APPLY.REMOTE))

  ERRORS     ((REMOTE.EVAL.ERROR 0 (ERRORN))
              (REMOTE.APPLY.ERROR 1 (ERRORN))
              (REMOTE.READ.ERROR 2 (ERRORN)))
)
```

**RELATED FILES**

The modules CHATSERVER-NS, COURIERDEFS, COURIEREVALSERVE, COURIERIMAGESTREAM, MONITOR, REMOTEPSW and NSTALK all define Courier servers and/or Courier type definitions.



---



---

**CROCK**


---



---

By: Kelly Roach

New Owner: Herb Jellinek (Jellinek.pa@Xerox.com)

CROCK sets up an analog face clock in the user's environment. To use, LOAD CROCK.LCOM and call (CROCK). CROCK requires that PROCESSWORLD be running (automatic in Fugue or later).

**CROCK**

Function CROCK has the form

(CROCK *REGION*) [Function]

The first invocation creates a clock window, CROCKWINDOW, occupying REGION with style CROCK.DEFAULT.STYLE. If REGION is left NIL, a region will be prompted for. Subsequent invocations use CROCKWINDOW. Only one clock window may exist at any given time. The clock is updated once a minute.

**STYLE**

The clock's style is maintained as a property list and can be found by (WINDOWPROP CROCKWINDOW 'STYLE). There are four independent boolean properties which the user may control: HANDS (the hands of the clock), TIMES (time digits printed where the hands end), RINGS (rings on the clock face), and NUMBERS (12 numbers around the outside of the clock face). The style first used will be CROCK.DEFAULT.STYLE (bound to '(HANDS T TIMES NIL RINGS NIL NUMBERS T) when CROCK is first loaded).

**CROCK.DATEFORMAT**

The user can control how the date will be printed in CROCKWINDOW. CROCK.DATEFORMAT should have the form (DATEFORMAT . <tokens>) where each <token> is one of NO.DATE, NO.TIME, NUMBER.OF.MONTH, YEAR.LONG, SLASHES, SPACES, NO.LEADING.SPACES, TIME.ZONE, or NO.SECONDS. These are all listed on pp23.57-58 of the IRM. Unfortunately, some other possibilities, such as DAY.OF.WEEK have not been implemented by Interlisp-D yet and are therefore not available to CROCK yet. The default value for CROCK.DATEFORMAT is (DATEFORMAT NO.SECONDS). For example,

```
(SETQ CROCK.DATEFORMAT
  '(DATEFORMAT SLASHES NUMBER.OF.MONTH NO.SECONDS))
```

would make CROCK print a date string like

```
28/09/84 14:53
```

instead of a date string like

```
28-Sep-84 14:53
```

Since CROCK updates itself only once a minute, it is probably a good idea to always include NO.SECONDS in your CROCK.DATEFORMAT.

### CROCK.ALARM AND CROCK.TUNE

The user can set CROCK's alarm via

(CROCK.ALARM *DATESTRING*) [Function]

where DATESTRING is any arg acceptable to Interlisp's IDATE (such as the date CROCK prints in CROCKWINDOW). CROCK will act appropriately when time reaches DATESTRING. Dandelion users can set global CROCK.TUNE to a tune to be played by Interlisp's PLAYTUNE when CROCK's alarm acts.

### RECOMMENDED USAGE

The simplest way to call CROCK from your init file or other function is to set your CROCK globals, then call CROCK:

(SETQ CROCK.DEFAULT.STYLE *STYLE*) [Variable]

(SETQ CROCK.DATEFORMAT *DATEFORMAT*) [Variable]

(SETQ CROCK.TUNE *TUNE*) [Variable]

(CROCK *REGION*) [Function]

You supply <style>, <dateformat>, <tune>, and <region>. You only need the SETQs if you want non-default values. If no <region> is supplied, CROCK will prompt for one.

### LEFT MOUSE BUTTON

Buttoning CROCKWINDOW with the left mouse button requests immediate update of the clock. (Of course, it may take a while for the process scheduler to get to it.)

### MIDDLE MOUSE BUTTON

Buttoning CROCKWINDOW with the middle mouse button presents a menu of commands for modifying the clock's style. Menu item SHOW.STYLE prints the clock's style.

### RIGHT MOUSE BUTTON

Buttoning CROCKWINDOW with the left mouse button presents the usual window menu. RESHAPEing the CROCKWINDOW causes the clock to change its size to fit the new window region. CLOSEing the CROCKWINDOW deletes the clock process.

---



---

**DATEFORMAT-EDITOR**


---



---

By: Johannes A. G. M. Koomen  
 (Koomen.wbst@Xerox.com or Koomen@CS.Rochester.edu)

This document last edited on February 19, 1987.

### DESCRIPTION

DATEFORMAT-EDITOR provides a menu-based interface for creating and editing date formatting lists (see IRM, Section 12.5). The menu is a Free Menu (see FREEMENU in Medley Release Notes), and looks like:

```

Date Format Editor
Quit  Abort

DATE: dd-mon-yy  none
      dd/mon/yy  dd mon yy  mon dd, yy
Year: short long
Month: alpha-short alpha-long numeric
Weekday: none long short
Leading spaces: yes no
TIME: hh:mm:ss hh:mm none
Time Zone: no yes
  
```

### INTERFACE

(EDIT-DATEFORMAT *DATEFORMAT*)

[Function]

DATEFORMAT is either NIL or the value returned from a call to the function DATEFORMAT (see IRM, Section 12.5). EDIT-DATEFORMAT starts by pre-selecting date formatting keys according to DATEFORMAT, or default ones if DATEFORMAT is NIL. It then enters a busy-wait loop, blocking until the DateFormat Editor window is closed, or **Quit** or **Abort** is selected. EDIT-DATEFORMAT returns a new value obtained from the function DATEFORMAT given the selected date formatting keys if **Quit** was selected, otherwise NIL.

DATEFORMAT-EDITOR-ITEMS

[Variable]

A list of items acceptable to the function FM.FORMATMENU (see FREEMENU in the Release Notes). Unfortunately, some of the date format details are embedded in the DateFormat Editor, rather than in these items, so leave ID's and LABEL's alone, otherwise mung around to your heart's content if you desire a different layout for the DateFormat Editor. Initial value is reflected by the screen snap above.

(GET-DATEFORMAT-EDITOR *RECOMPUTE?*)

[Document Object]

Returns the FreeMenu window of the DateFormat Editor. If *RECOMPUTE?* is non-NIL, recomputes the FreeMenu. Use this function with argument T if you change the variable DATEFORMAT-EDITOR-ITEMS.

#### **EXTENDED DATEFORMAT OPTIONS**

MONTH.LONG

[DateFormat Option]

Provides for full names of months rather than the first three characters. For instance, "20 February 1987" is produced by (GDATE NIL (DATEFORMAT MONTH.LONG YEAR.LONG SPACES NO.TIME)).

MONTH.LEADING

[DateFormat Option]

Causes the month to appear before the day. For instance, "February 20, 1987" is produced by (GDATE NIL (DATEFORMAT MONTH.LEADING MONTH.LONG YEAR.LONG NO.TIME)). MONTH.LEADING implies SPACES and disables NUMBER.OF.MONTH.

---

---

**DEFAULTSUBITEMFN**

---

---

By: Nick Briggs (Briggs.pa@Xerox.com)

The DEFAULTSUBITEMFN module redefines the DEFAULTSUBITEMFN to permit an extended specification of menu subitems. If the CAR of the 4th element of a menu item is the keyword EVAL, the CADR of that 4th element is evaluated and the results used as the subitem specifications. During the evaluation the variables MENU and ITEM are bound respectively to the menu and item of which the EVAL subitem spec is a part. This module is only a stopgap measure until it is possible to easily redefine the BackgroundMenu subitem function, but it will provide this facility on all menus that do not explicitly specify a subitem function.

example menu item entries:

```
(foo foo.selected "No help for you!" (EVAL dynamic.foo.subitems))
```

using a variable containing subitems, or

```
(foo foo.selected "No help for you!" (EVAL (compute.foo.subitems)))
```

using a function to recompute the subitems.

It is prudent to make the expressions used in the EVAL subitems quite efficient, since they will be called many times.

---

---

## DIGI-CLOCK

---

---

By: Keith Mountford (Mountford.AISNorth@Xerox.Com)

### INTRODUCTION

DIGI-CLOCK is a digital clock which allows you to keep track of the time in multiple time zones.

### STARTING DIGI-CLOCK

Loading DIGI-CLOCK will kill any existing DIGI-CLOCK process and restart the clock. Once the clock is loaded it can be restarted by typing (DIGI-CLOCK) or (DIGI-CLOCK T). The second of these restarts the clock from scratch, rebuilding everything; the first, simply restarts the process and does not undo any changes made to the clock. Left buttoning in the window causes the clock to update itself. The clock updates itself approximately once a minute.

### CHANGING DIGI-CLOCK

The clock font, the time, the local time zone, the alarm, the alarm mode (loud or quite), the clock mode (12 or 24 hour) are all settable from the middle button menu. This menu also allows you to add clocks for other time zones. The auxilliary clocks also have middle button menus which allow you to set the time zone for that window and edit the time zone heading. The default is for all of the auxilliary clocks have the same font and changing the font in one changes the font in all of them unless the submenu item "Set Aux Clock Font In Just This Window" is selected. Selecting "Shape to Fit" will reshape the clock windows to their minimum size.

If the menu font options are not sufficient you can set the global variables \*DC-FONT\* and \*DC-AUXW-FONT\*. The date format is bound to the variable \*DC-DATEFORMAT\* and can be changed by editing or setting this variable. The clock does not deal with seconds gracefully in 12-hour mode and it will not allow NUMBER.OF.MONTH in 12-hour mode. The regional time zones are stored on the global list \*DC-TIME-ZONE-LIST\*.

### SETTING DIGI-CLOCK

Choosing "Set Time" from the middle button menu, brings up a menu which allows you to set the time.

### SETTING THE DIGI-CLOCK ALARM

DIGI-CLOCK includes an alarm clock which can be set to any number of dates in any order. The alarm stores a brief message to remind you why the alarm was set. To set the alarm choose the "Set Alarm" middle button option. Once you have set the time, the clock will prompt you for a message. This message can be longer than the window, but only one line long. When the alarm rings, the window will shape to fit the message.

The alarm calls the function RINGBELLS once a minute until the alarm is turned off, which can be annoying. To run the alarm in quiet mode, select Quiet Alarm from the middle button menu. Selecting Quiet Alarm changes this menu option to Loud Alarm and sets the alarm to run in quiet mode. Selecting Loud Alarm will toggle the alarm back to its original noisy setting.

To unset the alarm, select "Delete Alarm Setting" from the middle-button menu and then select the time you want deleted from the pop-up menu.

To turn the alarm off, select "Turn Alarm Off" from the middle-button menu.

---



---

## DOC-OBJECTS

---



---

Johannes A. G. M. Koomen  
(Koomen.wbst@Xerox.com or Koomen@CS.Rochester.edu)

Uses: TEDIT, IMAGEOBJ, DATEFORMAT-EDITOR

This document last edited on October 27, 1987.

### DESCRIPTION

DOC-OBJECTS is a generic, extensible interface for including image objects in TEdit documents. It hooks into TEdit by an extra entry on TEdit's middle button menu, as well as by redefining what happens on typing CTRL-O. Clicking the menu entry or typing CTRL-O brings up an *Objects* menu. Selecting an object causes an instance of the designated object to be inserted in the document at the position of the caret. Clicking outside the Objects menu has no effect. DOC-OBJECTS comes with a set of predefined Document Objects, which are described below. Additional Objects can easily be added to the Objects menu.

### Predefined Objects

**Time Stamp** [Document Object]

A TimeStamp reflects the date the document containing it was last PUT into a file. Each PUT causes a TimeStamp to be updated. Clicking the middle button over a TimeStamp brings up a DateFormat editor. The TimeStamp can be given any appearance consistent with the function DATEFORMAT (see IRM, Section 12.5). The object following the next colon is a TimeStamp object for this file: 15 Sep 88 18:11 PDT (Thursday) Individual characters of a TimeStamp cannot be altered by TEdit, but a TimeStamp can be given arbitrary TEdit Looks. The DATEFORMAT-EDITOR package is automatically loaded by the DOC-OBJECTS package.

**File Stamp** [Document Object]

A FileStamp reflects the name of the file into which the document containing it was last PUT. Each PUT causes a FileStamp to be updated. It cannot be edited. A FileStamp is initially displayed as "-- not yet filed --".

**Include** [Document Object]

This document object is a dynamic version of the static TEdit Include command, and is intended to facilitate the unbundling of document chapters and sections, while maintaining the ability to print the entire document or any portion of it. When an Include object is created, the user is prompted for a file name. An Include object can be enabled or disabled. If it is enabled, the object shows in the TEdit window as '@Include[MySubFile.TEdit]', and the indicated file will be included during a hardcopy operation. If it is disabled, then the object shows (both in the TEdit window and on hardcopy) as '@DoNotInclude[MySubFile.TEdit]'.

Middle-clicking on an Include object pops up a menu with the following fields: "New File" (prompt for a new file name), "Edit File" (TEdit the Include file, or bring it to the top if it is already being



edited), "Enable" (include the file during hardcopy), and "Disable" (do not include the file during hardcopy).

Two caveats:

- 1) For best results, make an Include object the last thing in a paragraph, or put it in a paragraph of its own, and set the line and paragraph leadings to 0. The Include object forces a paragraph break right after the Include object during hardcopy, to prevent the looks of the paragraph containing the Include object to mask the looks of the first paragraph in the file being included.
- 2) A document containing Include objects is best hardcopied from a FileBrowser window, rather than through the hardcopy command on the TEdit window menu. It will work properly either way, but it's a bit unnerving to watch TEdit trying to reflect on the display the inclusion of one or more files before hardcopy and the removal of the included files after hardcopy.

Horizontal Rule

[Document Object]

This provides a more user-friendly interface to the HRULE package (which is automatically loaded by the DOC-OBJECTS package). Upon selecting it a numberpad is brought up repeatedly, with which the user can indicate the thickness of alternating black and white lines. The resulting HRule object is inserted in the document whenever the numberpad is aborted or returns 0. The DOC-OBJECTS package also modifies the HRule object such that it can be edited: clicking the middle button over an HRule object brings up a structure editor (such as SEdit) on a list containing the thicknesses of the lines composing the HRule. This list can be altered in any way, as long as the editor returns another list of numbers (presumably of odd length).

Eval'd Form

[Document Object]

Selecting this object causes a type-in window to pop up. The value of the form typed in is assumed to be an image object. This is what TEdit used to do on typing CTRL-O. For TEdit's purpose, an image object is a Lisp value of type IMAGEOBJ, BITMAP, STRINGP, LITATOM, or REGION. The latter is assumed to refer to a region of the screen.

Screen Snap

[Document Object]

Selecting this object prompts for a region of the screen. A bitmap containing a copy of the given region of the screen is inserted in the document. This is equivalent to clicking the right button in the display's background while holding the SHIFT key down.

### Extending the Document Objects interface

DocObjectsMenuCommands

[Variable]

This variable contains a list of menu items which are displayed in the Document Objects pop-up menu. It is analogous to the variable BackgroundMenuCommands (cf. IRM, Section 28.8). The Lisp form in each item is assumed to evaluate to an image object (as defined under Eval'd Form described above).

DocObjectsMenu

[Variable]

This variable caches the Document Objects menu. Set it to NIL whenever you alter the variable DocObjectsMenuCommands.

**DocObjectsMenuFont** [Variable]

This variable contains a font descriptor which is used for displaying the items in the Document Objects menu. The initial value is (FONTCREATE '(MODERN 12 BOLD)). Set the variable DocObjectsMenu to NIL whenever you alter DocObjectsMenuFont.

**(DOCOBJ-STRING-IMAGEBOX STRING IMAGESTREAM)** [Function]

A useful function for Document Objects that wish to display as a string of characters (such as a TimeStamp). The Document Object's IMAGEBOXFN can call this function to obtain an image box with the TEdit Looks that apply to the Document Object taken into account.

**(DOCOBJ-WAIT-MOUSE WINDOWSTREAM)** [Function]

A useful function for Document Objects that wish to assure that their buttoneventfn takes action only if the mouse buttons were let up within the Object's region (i.e., the clipping region of the Object's window stream). It returns T when the mouse buttons go up within the region, or NIL when the mouse moves out of the region while a button is still down.

#### **Future predefined Document Objects**

Watch this space for objects such as Index & Index Entry, Citation & Bibliography, ...

---

---

**DONZ**

---

---

**The *Excruiatingly* User Friendly Environment**

By: Jeff Shrager et al.

Checked out for Medley by Larry Masinter (Masinter.PA@Xerox.COM)

Files: DONZ. DONZ.dfasl DONZ.TEDIT

**Description:**

Loading DONZ starts a background process (DONZ.RUN) which causes your ICONS to become "user friendly". Telling you what this means would spoil all the fun of discovery.

**Customizing DONZ:**

The delay between activations of DONZ is done by (DISMISS DONZ.DELAY). Its value defaults to 5000 (5 seconds). When DONZ wakes up, it tries out one window, selected at random, from all the windows in (OPENWINDOWS). If the selected window is not an ICON, nothing happens and DONZ wakes up again in 5 more seconds. If that window is an ICON, then DONZ tries to find a message for it as described below. This method results in DONZ's "friendliness rate" running approximately in proportion to the ratio of icons to opened windows on your screen. Thus, if you are doing real work, DONZ won't bug you, but if you just have a screen full of icons, DONZ will be *exceedingly friendly*.

The list DONZ.TEST.MESSAGE.ALIST has the form:

(frob1 frob2 frob3...)

where each frob is of the form:

(testfn msg1 msg2 msg3...)

TESTFN will be called with one argument: the \*MAIN\* window with which this icon is associated. That is, for instance, the TEDIT window that the icon will expand to...NOT the icon window. TESTFN should decide whether or not this window is of a type that it will handle, and return T or NIL as appropriate. When one of the TESTFNs returns T, one of the messages in the tail of the associated frob list will be selected at random and displayed...in the way that they are displayed...you'll see! If none of the frobs accepts responsibility for the present icon, there are a few default messages built into DONZ.

The msgs should be lists containing individual words (appropriately capitalized) so that .PARA in the PRINTOUT can do the appropriate word division.

**Notes:**

DONZ has to be killed via the PSW. Be gentle.

DONZ continues to run during screen idle. This results in fairly funny theatrics on the part of your icons.

The display is done with PRINOUT which does some bogusness to indent, erasing some of the good parts of the window. This is slightly messy, but otherwise innocuous.

Anyone who can guess the origin of the name of this package belongs on the East Coast.

**Acknowledgements:**

Thanks to Ros Chast for originating the idea, Mike Kazar and Dave Nichols for their original implementation of DONZ at CMU.

---



---

## DSPSCALE

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

DSPSCALE allows a program to output to different types of streams (display, Interpress, etc.) without corrections for scaling. This module provides self-scaling graphics through two different methods: a virtual self-scaling *image stream* that overlays a regular image stream and/or new versions of the various image stream graphic manipulation functions (DRAWLINE, DSPTOPMARGIN, etc.). The goal of both methods is to make it possible to modify the normal scaling factor of an image stream without modification to the program generating the output.

### VIRTUAL SELF-SCALING IMAGE STREAM

This module implements a virtual image stream type, called SCALED, which is used to overlay any other image stream and provide automatic scaling to the natural scale of the image stream or any user selected scaling factor. The function OPENIMAGESTREAM is used to overlay a scaled image stream over a regular one. For example, the following will open a scaled image stream on top of an Interpress image stream:

```
(OPENIMAGESTREAM (OPENIMAGESTREAM 'TEST.IP 'INTERPRESS) 'SCALED)
```

The only difference between the virtual stream and a normal image stream is that the SCALE argument to the DSPSCALE function is active and can be used to change the scale of the stream (multiplying the scale specified by the standard scaling factor of the stream).

### SELF-SCALING GRAPHICS FUNCTIONS

As an alternative to the self-scaling image stream, self scaling versions of the various graphics functions are provided. For most of the graphic functions, self-scaling versions have been defined which have an ! (exclamation point) at the end of their name (eg. DRAWLINE vs. DRAWLINE!):

CENTERPRINTINREGION!	DRAWELLIPSE!	DSPSCALE!
CHARWIDTH!	DRAWLINE!	RELDRAWTO!
CHARWIDTHY!	DRAWPOINT!	RELMOVETO!
CURSORPOSITION!	DRAWPOLYGON!	SCALEDBITBLT!
BITBLT!	DRAWTO!	STRINGREGION!
BITMAPBIT!	FILLCIRCLE!	STRINGWIDTH!
BLTSHADE!	FILLPOLYGON!	DSPSPACEFACTOR!
DSPBACKUP!	FONTPROP!	DSPTOPMARGIN!
DSPBOTTOMMARGIN!	GETPOSITION!	DSPXOFFSET!
DSPCLIPPINGREGION!	DSPLEFTMARGIN!	DSPXPOSITION!
DRAWARC!	DSPLINEFEED!	DSPYOFFSET!
DRAWBETWEEN!	MOVETO!	DSPYPOSITION!
DRAWCIRCLE!	MOVETOUPPERLEFT!	
DRAWCURVE!	DSPRIGHTMARGIN!	

The set includes both output *and* input functions since it is necessary when getting, for example, a mouse position, to unscale the position to put it back into the program's virtual coordinate system. By default, these functions can be used directly in place of their non-! counterparts and they will automatically scale their arguments to the DSPSCALE of the output stream. Some of the above functions are not identical with their non-! counterparts, as explained below:

(DSPSCALE! SCALE STREAM) [Function]

In this version of DSPSCALE, the *SCALE* argument is active and will multiply *STREAM*'s normal scaling factor. If you have a program that draws a circle, for example, to a window using the appropriate ! functions, you can cause it to draw a different size circle (larger or smaller) by using DSPSCALE! to change the scaling factor of the window without touching the source program.

(CHARWIDTH! CHARCODE FONT STREAM) [Function]

(CHARWIDTHY! CHARCODE FONT STREAM) [Function]

(FONTPROP! FONT PROP STREAM) [Function]

(STRINGWIDTH! STR FONT FLG RDTBL STREAM) [Function]

All of the above functions have one extra argument (as compared to their non-! equivalents) which is the *STREAM* in question. This is necessary to do the scaling calculations.

The module also defines a couple of new stream manipulation functions:

(DSPTRANSLATE! Tx Ty STREAM) or (DSPTRANSLATE! POSITION STREAM) [Function]

Defines the amount of X and Y translation that should be added to graphic operations to *STREAM*. Similar to DSPTRANSLATE but works even if the image stream does not have an IMTRANSLATE method. The second form of the arguments is for backward (Koto) compatibility.

(DSPUNITS! UNITS STREAM) [Function]

Essentially the inverse of DSPSCALE!, this function lets you set how many *UNITS* (pixels or whatever) the source program generates for each unit pixel on the output stream (multiplied by the output stream's default scaling).

It is possible to use both the virtual image stream and the self-scaling graphics functions together as long as the self-scaling graphics functions are applied to the real stream, not the virtual one.

### Lisp Data Type Scaling Functions

The routines below are used by the ! functions and the virtual image stream for scaling numbers, positions, regions and other data types and are useful for defining other self-scaling functions:

(DSPSCALE.BRUSH BRUSH STREAM) [Function]

(DSPSCALE.DASHING DASHING STREAM) [Function]

(DSPSCALE.POINTS KNOTS STREAM) [Function]

(DSPSCALE.REGION REGION STREAM [SmashRegion]) [Function]

(DSPSCALE.NUMBER NUMBER STREAM) [Function]

(DSPSCALE.POSITION POSITION STREAM [SmashPosition]) [Function]

(DSPSCALE.XPOSITION NUMBER STREAM) [Function]

(DSPSCALE.YPOSITION NUMBER STREAM) [Function]

(DSPSCALE.WIDTH WIDTH STREAM) [Function]

(DSPUNSCALE.REGION REGION STREAM [SmashRegion]) [Function]

(DSPUNSCALE.POSITION POSITION STREAM [SmashPosition]) [Function]

(DSPUNSCALE.NUMBER NUMBER STREAM [OFFSET]) [Function]

(DSPUNSCALE.XPOSITION NUMBER STREAM) [Macro]

(DSPUNSCALE.YPOSITION NUMBER STREAM) [Macro]

**EDITBG**

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

EDITBG is a tool for editing both the background and background border shades. The functions CHANGEBACKGROUND and CHANGEBACKGROUNDDBORDER both take a shade argument but the shade is interpreted differently. A normal black & white shade consists of 16 pixels (see EDITSHADE in the Interlisp Reference Manual) as does the border shade, which covers twice the area. The normal shade has 4 x 4 pixels but the border shade has 2 x 8 pixels where the pixels are twice as tall. WHITESHADE and BLACKSHADE appear the same for both, as does the standard background shade (shown below) but arbitrary shades do not appear the same.

Background

	14	13	12
	10	9	8
7	6		4
3	2		0

Background Border

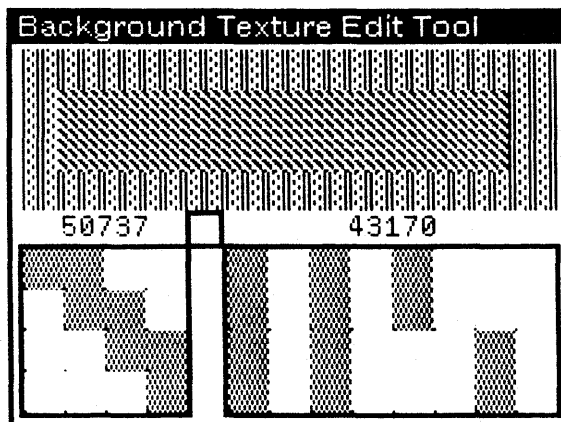
	14	13	12		10	9	8
7	6		4	3	2		0

$$34850 = 2 \uparrow 15 + 2 \uparrow 11 + 2 \uparrow 5 + 2 \uparrow 1$$

(EDITBACKGROUND)

[Function]

Brings up an edit tool (also available from the background menu) which lets you edit both a normal shade and a border shade and see how they combine:



The bottom half of the window has a background texture editor on the left and a border texture editor on the right. The top half of the window shows the background texture within the border texture as it would appear on the screen. Buttoning the small box in the center of the window will change the background and border textures on the screen to those displayed.

---

---

**EDITKEYS**

---

---

By: Larry Masinter (Masinter.pa@Xerox.com)

- EDITKEYS provides a set of "logical" keys corresponding to the Dandelion (1108) function keys. This allows 1132 users to take advantage of interfaces designed for the 1108 keyboard. Calling (BUILDFNKEYS) builds a window with 8 keys like the one below:



These keys can be "pressed" by bugging the mouse (left or middle mouse button) inside the key's image. The effect of pressing one of these keys is to generate the same character code as the key generates on the 1108. The state of the shift keys (at the time the mouse is let up) are taken into consideration, but interfaces that use KEYDOWNP are not affected.



---

---

## EQUATIONS

---

---

By: Tad Hogg (Hogg.pa@Xerox.com)

### DESCRIPTION

This module provides interactive editing of mathematical equations within TEdit. An equation consists of a number of pieces of text and possibly one or more special symbols. For many purposes, such as deletion and copy selection, equations behave as a single (large) character in TEdit. Operations on their pieces are described below.

#### To load:

The equation editor and a standard set of equation types is obtained by loading EQUATIONS.LCOM.

#### To use:

This section describes the procedures by which equations can be inserted into documents and their pieces modified.

#### *Adding an equation to a TEdit document:*

To add an equation, first move the caret to the desired insertion point and then select "Equation" from the main TEdit menu (obtained by holding down the middle button in the window's title bar). This will display a menu of known equation types. Selecting one of these will insert the corresponding equation into the document at the current location of the caret. To abort the insertion, click outside the menu. Once the equation is inserted, a subeditor will be created for each of the equation pieces, one at a time. This can be used to fill in the various pieces of the equation and its use is described below. Some equation types will prompt for additional information before inserting the new equation (e.g. inserting a matrix will prompt for the desired number of rows and columns).

#### *Editing a currently existing equation:*

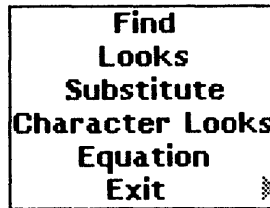
In order to modify a piece of an existing equation (e.g. the numerator of a fraction), the piece must be selected with the mouse in one of two ways. First, you can point at the piece in the displayed equation and press the left mouse button. Alternatively, pointing at the equation and pressing the middle button will display a menu from which the desired piece can be selected. This is useful for selecting pieces that are too small to conveniently point at with the mouse. In either case, if a piece is selected, a subeditor will start on that piece. In addition, some equation types may also allow changes to global properties when no specific piece is selected (e.g. changing the number of rows in a matrix).

#### *Using the equation piece subeditor:*

The subeditor is attached to the bottom of the main edit window and allows individual pieces of the equation to be modified with normal TEdit operations. While a subeditor is active, the corresponding piece of the equation in the main window is inverted. Since the text in equation

pieces must be on a single line, the subeditor will not accept control characters such as carriage returns. Instead the edit window will flash when such characters are typed.

In the subeditor, the TEdit menu is modified to provide a limited set of TEdit commands as well as additional commands relevant to equations. The menu appears as



Selecting *Find*, *Looks*, *Substitute* or *Character Looks* invokes the corresponding TEdit action. Selecting *Equation* acts as described above and allows equations to be embedded inside other equations. *Exit* ends the subedit of the equation piece, updates the equation in the main editor and, if this is a newly inserted equation, automatically starts editing the next piece.

The *Exit* item also has three possible subitems which are used to exit from the equation editor and specify a desired follow up action. Specifically, *Next Piece* ends the edit of the current equation piece and creates a new editor on the next piece. In this context, the pieces of the equation are considered to form a circular list so that successive uses of the *Next Piece* option will edit each piece of the equation in turn. The second subitem, *Finish Eqn*, ends the current equation edit and does not continue with any other pieces of the equation. Finally, *Abort* ends the current edit without changing the equation.

When a subeditor is terminated, any TEdit looks or formatting other than character fonts and sub/superscripting are ignored.

The subeditor can also be terminated by the key normally used to advance to the next fill-in slot in TEdit (i.e. text of the form ">>...<<"). Specifically, if there are no remaining slots in the subeditor, using this key is equivalent to selecting *Exit* from the command menu described above. By default, this key is the middle-blank key on Dolphins and Dorados and the OPEN key on DLions.

#### User Switches:

The global variable *EquationFontSpecs* is an array of font specifications in order of increasing size which is used to determine initial fonts for the equation pieces. This can be modified if additional or different default fonts are desired.

The global variable *EQ.UseNSChars* determines the kind of characters to use when displaying equations that use special symbols (e.g. sum or product) on the screen. Specifically, if non-NIL then symbols from the NS character set are used, otherwise the Sigma 20 font is used. It is initially set to NIL.

When NS characters are used (on the screen when *EQ.UseNSChars* is non-NIL, or for Interpress), the global variable *EQ.NSChars* determines the particular NS characters to use. It is a property list of the form (TYPE1 ITEM1 ...) where TYPE is the kind of equation (e.g. SUM, PRODUCT, etc) and ITEM gives the font and character number to use, e.g. ((MODERN 30) 61301) for an INTEGRAL. This variable compensates for the lack of large symbols in various Interpress fonts.

The file EQUATIONPROGRAM.TEDIT describes how to define new kinds of equations, as well as how to create equations in a program with function calls.

**Examples:**

Examples of equations are given in the file EQUATIONEXAMPLES.TEDIT. The equation module must be loaded before reading this file.

**Limitations:**

- Equations that are larger than the available room in the current TEdit window will not be displayed.
- The text of each piece of an equation must be on a single line.
- All image objects inserted into equations, as well as the equations themselves, must not have any kerning, i.e. the XKERN field of all imagebox records must be zero.

**Koto Incompatibility:**

Due to a change in image object I/O, files containing equations written in Lyric/Medley may not be readable in Koto.

---



---

**EQUATIONEDITOR**


---



---

**PROGRAMMER'S GUIDE**

By: Tad Hogg (Hogg.pa@Xerox.com)

**DESCRIPTION**

This document describes how to define new kinds of equations to be used with the equation editor in TEdit, and how to construct equation image objects by function calls.

**User Switches:**

The global variable *EquationDefaultSelectionFn* specifies the default function to be called when an equation is selected with the middle mouse button. The initial value, EQIO.DefaultSelectFn, allows the user to select a piece of the equation by selecting from a menu.

The global variable *UnknownEquationData* is a formatted string to use for displaying equations whose types are not defined.

The global variable *EquationInfo* is used to record all currently defined equation types. The specification information can be obtained by calling

(EQIO.GetInfo *type info*) [Function]

which returns the specified info for equations of the given type. Any of the PROPS mentioned below for EQIO.AddType, or *formFn* or *numPieces*, can be used as a value for *info* to get the corresponding data for this type of equation. Example: (EQIO.GetInfo 'fraction 'numPieces) returns the number of pieces in a fraction.

Individual specification items of an existing equation type can be modified using

(EQIO.SetInfo *type info newValue*) [Function]

although the caller must be sure that any new information is consistent with the remaining properties. For example, (EQIO.SetInfo 'fraction 'menuLabel myLabel) will make the value of myLabel be used for fractions in the equation type menu.

**Defining new kinds of equations:**

This module allows new types of equations to be defined. An equation consists of some number of pieces of text and, perhaps, some extra symbols or lines. A method for computing the relative position of the various pieces must be specified when new equation types are used. The pieces of equations consist of "formatted strings" which allow font information to be associated with strings and can also include image objects (which thus allows equations to contain other equations). Formatted strings are described below. Additional properties can be specified to determine the particular behavior of the new equation.

Additionally, a function can be provided to allow equations of the new type to be created under program control. Typically these are named EQ.Make.xxx where xxx = atom specifying the equation type.

Specifically, a new equation type (or a new definition for an existing type) is created by calling

(EQIO.AddType *type formFn numPieces PROPS*) [Function]

where

- *type* is an atom identifying the equation type (e.g. fraction)
- *formFn* is the name of a function, described in detail below, which specifies the relative location of the equation pieces and, if requested, draws any extra lines or symbols required by the equation that are not included in any of its pieces. The *formFn* can also specify the selection region to be used for each piece of the equation, i.e. that region of the equation within which the left mouse button can be used to select the piece.
- *numPieces* is the number of parts the equation has (e.g. a fraction has two parts: numerator and denominator). If the equation has a variable number of pieces, then *numPieces* is the default initial value.
- *PROPS* is a prop list of optional properties for the equation which are described below.

*The equation form function:*

The form function is called with arguments (*eqnObj imageStream draw?*) and specifies the size of the entire equation and the location of each piece with respect to the lower left corner of the box. Furthermore, if *draw?* is non-NIL, it draws any extra lines or symbols required by the equation that are not included in any of its pieces. (If *draw?* is NIL, nothing should be drawn -- the function is being called only to determine how big the equation is and the relative location of its parts.) The *formFn* can also specify the selection region to be used for each part of the equation.

For example, a fraction has two parts (numerator and denominator) and a single extra line between them. In this case the *formFn* draws the line and specifies the location of the numerator and denominator.

Specifically, the *formFn* should return an equation specification created by a call to

(EQIO.MakeSpec *box dataSpecList*) [Function]

where *box* is an IMAGEBOX which specifies the size of the equation; and *dataSpecList* is a list containing a piece specification for each piece of the equation. A piece specification is created by a call to

(EQIO.MakeDataSpec *pos selectRegion*) [Function]

where *pos* is the position, relative to the lower left corner of the box, where this piece is to be displayed and *selectRegion* gives the selection region to use for this piece (relative to the l.l. corner of the equation box). If *selectRegion* is NIL, then the region within which the piece is displayed will be used as the selection region. Normally the *selectRegion* should include the display region inside it, and is intended to allow selection regions to be larger than just the display region. Note that *pos* specifies where the stream should be positioned before displaying the formatted string defining the contents of the piece.

The ATTACHEDBOX routines, described below, may be useful for constructing the form functions of new equations since it provides functions to position various regions so that they do not overlap.

*Equation type properties:*

The following properties can be specified for any equation type when it is defined with EQIO.AddType. Note that all equations of a given type have the same values for these properties.

- **changeFn** A function with argument (eqnObj) called when the number of pieces in a variable piece equation is changed. It is meant to allow any specific properties such as saved menus to be adjusted to reflect the change.
- **initialData** A way to specify the text to be used when equations of this type are created. It can be either the name of a function or a list. If it is a list, then it should contain a single integer for each piece of the equation. This number specifies how the font size for that piece should be changed relative to the initial font size set in TEdit. For example, 0 means use the default font, + 1 means use the next bigger font, -2 means use a font two sizes smaller, etc. Missing values default to zero, i.e. the pieces are initially set in the normal size font. The actual fonts used are specified by the array EquationFontSpecs and any request for a font larger (smaller) than the largest (smallest) available in the array defaults to using the largest (smallest). In this case, the initial text will be a single blank.

If initialData is a function, it is called with arguments (initialFontSpec type numPieces dataList) when a new equation of this type is created. It should return a list of formatted strings, with each item in the list to be used for the corresponding piece of the equation. The argument initialFontSpec will specify the current font when the equation is added; and numPieces will be the number of pieces in the equation. dataList, if non-NIL, is a list of items to use for each of the pieces of the equation. The items can be either format strings, or just a string in which case the initialData function should attach an appropriate font.

- **initialPropFn** A function with argument (type) called when a new equation of this type is created. It returns a prop list to be used as properties for this equation. These values are added to (and override) any props given in objectProps. For example, this allows the user to specify the number of rows and columns in a new matrix. If the number of pieces is to be specified, it should be returned as the value of the numPieces prop, and the equation type should allow a variable number of pieces.
- **makeFn** A function which constructs an image obj for this type of equation from its arguments. Generally this will just provide a convenient way of calling EQN.Make.
- **menuLabel** A label to use for this type in the equation type menu displayed after selecting Equation in the main TEdit menu. If not given, the type atom is used as the label.
- **objectProps** A prop list of properties and their initial values that are needed for this equation. These properties can be used by the formFn to lay out the equation and will typically be modified by the wholeEditFn.
- **pieceNames** A list of names of the pieces of the equation. This is used by the default middle button selection function to provide a menu of choices. E.g. ("numerator" "denominator") for a fraction. If this property is not specified, then the default menu will just contain the numbers of the pieces. This is generally meant for equations with a fixed number of pieces.
- **specialSelectFn** A function with argument (eqnObj) to be called when the equation is selected with the middle button instead of the default action. It should return the number of the piece selected, or NIL if no piece of the equation is selected. The selected piece, if any, will then be edited.

- *wholeEditFn* A function with arguments (*eqnObj* *window* *button*) called when the entire equation, rather than a single piece, is selected. This can be used to change global properties of the equation and should return non-NIL if the object is modified, NIL otherwise. *window* is the window which contains the equation and *button* is the mouse button used to select it.
- *variable?* Non-NIL to indicate this equation type allows a variable number of pieces.

#### Equation data functions:

The functions used with new equation types should make use of the routines provided for formatted strings as well as the following functions when using the various equation data:

- (EQIO.EqnType *eqnObj*) returns the atom specifying the kind of equation *eqnObj* is.
- (EQIO.EqnDataList *eqnObj*) returns the list of formatted strings which specify the pieces of the equation.
- (EQIO.SetDataList *eqnObj* *newDataList*) replaces the data list (i.e. the list of formatted strings corresponding to each piece of the equation) with *newDataList*. The caller must update the number of pieces in the equation appropriately. This is useful, for instance, when the *wholeEditFn* has made major changes in the equation.
- (EQIO.EqnData *eqnObj* *piece#*) returns the formatted string corresponding to the piece of *eqnObj* specified by *piece#*.
- (EQIO.EqnProperty *eqnObj* *prop* {*newValue*}) returns the current value of the specified property of *eqnObj* if *newValue* is not present, otherwise sets the specified property to the value of *newValue* even if it is NIL. This can be used to associate arbitrary properties with individual equations. Currently, the following properties are used by the equation editor and should not be used for other purposes:
  - *fontSpec* a specification of the current font at the time the equation was created
  - *numPieces* the current number of pieces in a variable-piece equation
  - *selectionMenu* the current default middle-button selection menu for a variable-piece equation

When equations are copied, any data items that are not atoms, strings or lists are set to NIL. These items are also PRIN2'ed on files. Thus other data types should only be used to cache values (e.g. menus) that can be recomputed if necessary from the other properties.

- (EQIO.NumPieces *eqnObj* {*newValue*}) returns the current number of pieces in *eqnObj* if *newValue* is not present. Otherwise if *eqnObj* is a variable-piece equation, it sets the number of pieces to *newValue* and adjusts any necessary properties by calling the equation's *changeFn*.

Additionally, properties can be associated with the equation type itself by use of

- (EQIO.TypeProp *type* *prop* {*newValue*}) which gets or sets the property *prop* for equation *type*. This can be used to save properties that are the same for all equations of a given type (e.g. selection menus for equations with a fixed number of pieces).

Equation image objects can be created by a program (and then, for example, inserted into TEdit) by use of

- (EQN.Make type dataList fontSpec PROPS) which makes a *type* equation whose arguments are format strings contained in *dataList*. *fontSpec* is an initial font specification and *PROPS* is a prop list of equation properties which should include *numPieces* for equations with a variable number of pieces.

## FORMATSTRINGS

The basic components of equations are represented as formatstrings which allow fonts and super/subscripting to be associated with strings and can also include image objects. A format string is a list of items, each of which is either an imageobject or a list giving a font specification, a string and an optional shift specifying the number of points to move up when displaying the string. The following functions can be used to create and display format strings as well as insert and extract them from TEXTSTREAMs. All formatstrings must be on a single line.

### Functions:

For creating and accessing format strings:

(FS.MakeItem *fontSpec string shift*) [Function]

creates a formatstring item from *fontSpec*, a font specification such as (Gacha 10), *string* and *shift*. The string can be null.

(FS.ItemFont *item*) [Function]

returns the font associated with *item*, or NIL if *item* is an imageobject.

(FS.ItemValue *item*) [Function]

returns *item* if it is an imageobject, otherwise its associated string.

(FS.ItemShift *item*) [Function]

returns the shift associated with *item*.

For inserting and extracting from TEXTSTREAMs:

(FS.Extract *stream*) [Function]

returns a format string created from the text in the TEXTSTREAM stream. Any unallowed characters (as determined by FS.AllowedChar) in the stream are ignored. Note that any format information other than character fonts, super/subscripting and image objects is discarded. The file pointer associated with the stream is modified.

(FS.Insert *data stream*) [Function]

inserts the format string *data* at the current location in TEXTSTREAM stream.

For displaying and manipulating the format strings:

(FS.Box *data imageStream*) [Function]

returns an IMAGEBOX specifying the size of the format string *data* on *imageStream*.

(FS.Copy *data*) [Function]

returns a copy of the format string *data*.



(FS.Display *data imageStream invert?*) [Function]

displays the format string *data* on *imageStream*. If *invert?* is non-NIL, the display is inverted.

(FS.Get *fileStream*) [Function]

reads a formatstring, or list of formatstrings, from the current location on *fileStream*.

(FS.Put *data fileStream*) [Function]

prints the formatstring (or list of formatstrings) *data* to *fileStream*.

Additional functions:

(FS.AllowedChar *charcode*) [Function]

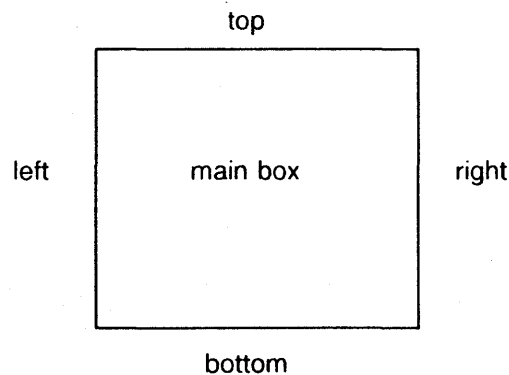
returns non-NIL if *charcode* is allowed in formatstrings.

(FS.RealStringP *item nullOK*) [Function]

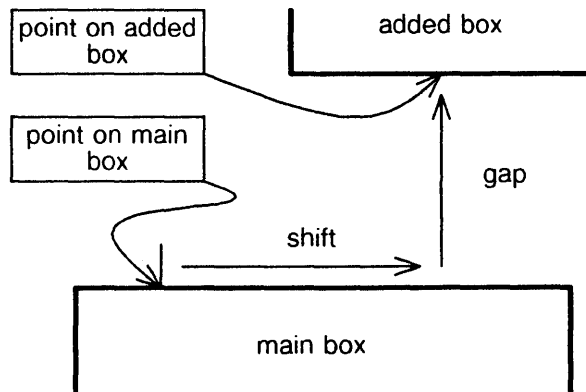
returns non-NIL if *item*'s value is a string (rather than an imageobject) and either *nullOK* is non-NIL or the string is not the nul string.

### ATTACHED BOXES

The following functions place image boxes in specific locations with respect to a main box so that the added boxes won't overlap. The desired position of a new box is specified by the side of the main box to place it next to and the position of a point on the side of the new box with respect to a point on the side of the main box. The placed regions are specified with respect to the lower left corner of the main box. Sides are specified by one of the atoms *top*, *bottom*, *left* or *right* and are with respect to the main box.



The position of the added box is specified relative to some side of the main box. Specifically, the location of a reference point on the near side of the added box, the *addPt*, is given with respect to a reference point on the side of the main box, the *mainPt*. The possible points along the side are specified by one of the atoms *low*, *high*, *center* or *display* corresponding to the corner nearest the lower left corner of the box, the corner farthest from the l.l. corner of the box, the center of the side, and the display point of the image box respectively. The location of the *addPt* with respect to the *mainPt* is specified by a distance along the side, the *shift*, and a distance perpendicular to the side, the *gap*. These distances can be positive or negative. A negative value for the *gap* will cause the added box to overlap the main box. All boxes are assumed to have no kerning (i.e. *XKERN* field is zero).



### Functions:

For creating and accessing format strings:

(AB.PositionRegion *mainBox addedRegions side mainPt addBox addPt gap shift clear*) [Function]

Positions *addBox* with respect to *mainBox* avoiding overlap with previously added regions. The parameters are: *mainBox* is an image box specifying the main box; *addedRegions* is a list of regions (measured with respect to the lower left corner of *mainBox*) that have already been placed next to the main box; *side* is the side (one of *top*, *bottom*, *left* or *right*) of the main box next to which the new box should be placed; *mainPt* is the reference point along the side of the main box (one of *low*, *high*, *center* or *display*); *addBox* is an image box specifying the box to be placed; *addPt* is the reference point along the side of the added box; *gap* and *shift* specify the relative positions of the reference points; and *clear* is the minimum (nonnegative) distance that the new box is allowed to be from any of the previously added regions (NIL defaults to zero which prevents any overlap of the added regions).

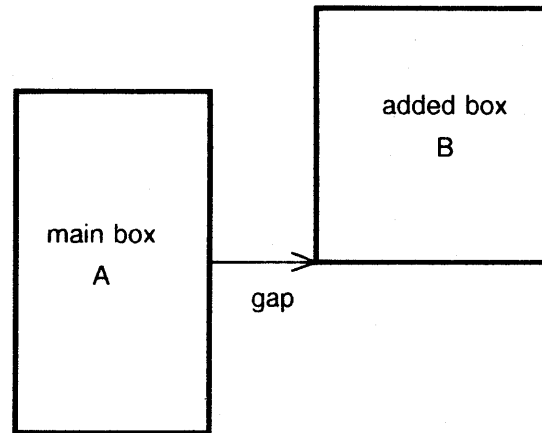
The function returns a list of the form (*region newAddedRegions*) where *region* is the region, w.r.t. the lower left corner of *mainBox*, where *addBox* was placed and *newAddedRegions* is the list of added regions updated to include this newly placed region. If the specified location of *addBox* causes it to be within a distance *clear* of any of the regions in *addedRegions*, the box is moved away from the main box in a direction perpendicular to the side (i.e. the gap is increased) until it is far enough from the previous regions.

(AB.Position2Regions *mainBox addedRegions side highBox highPt lowBox lowPt highGap lowGap highShift lowShift clear*) [Function]

This function places two boxes next to the same side of *mainBox*. If the two new boxes are within a distance *clear* of each other, they are moved apart in a direction parallel to the side next to which they are placed so that the distance each box moves is proportional to its size. Then these regions are individually checked for being too close to previously added regions and, if necessary, are moved away from the main box (i.e. perpendicular to the side). The function returns a list of the form (*highRegion lowRegion newAddedRegions*).

Example:

To place box B to the right of box A such that the low point of the near side of box B is a distance *gap* from the center of the right side of A, i.e.



use `(AB.PositionRegion A addedRegions 'right 'center B 'low gap 0)` where *addedRegions* is a list of previously added regions which should not overlap *B*.

---



---

**EQUATION EDITOR EXAMPLES**


---



---

By: Tad Hogg(Hogg.pa@Xerox.com)

**DESCRIPTION**

These are some examples of text formatted using the Equation Editor (contained in the file EQUATIONS).

**Examples:**

fraction:  $a = \frac{2x+5}{23}$

sum:  $s = \sum_{i=1}^m i = \frac{m(m+1)}{2}$

integral:  $\int_0^1 x \, dx = 0.5$

sub/superscripts:  $H_m^n = 24 = \text{KL} X_i^{23}$

root:  $\sqrt{x + 34y + z}$

maximum:  $\max_{x \text{ in } S} f(x)$

limit:  $\lim_{x \rightarrow 0} f(x)$

matrix:

$$a = \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 12 \\ 22 \end{bmatrix}$$

$$A = \begin{bmatrix} 23x+2 & 4 & 0 \\ -6 & 8x+4 & 1 \\ 0 & 0 & x \end{bmatrix}$$

$$\binom{n}{0} + \binom{n}{0} + \dots + \binom{n}{0} = 2^n$$

$$|n| \quad \{n\} \quad [n] \quad (n) \quad \langle n \rangle$$

$$\begin{bmatrix} n \\ 0 \end{bmatrix} \quad \left\{ \begin{matrix} n \\ 0 \end{matrix} \right\} \quad \left[ \begin{matrix} n \\ 0 \end{matrix} \right] \quad \left( \begin{matrix} n \\ 0 \end{matrix} \right) \quad \left\langle \begin{matrix} n \\ 0 \end{matrix} \right\rangle$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \left\{ \begin{matrix} x \\ y \\ z \end{matrix} \right\} \quad \left[ \begin{matrix} x \\ y \\ z \end{matrix} \right] \quad \left( \begin{matrix} x \\ y \\ z \end{matrix} \right) \quad \left\langle \begin{matrix} x \\ y \\ z \end{matrix} \right\rangle$$

---



---

## ETHERBOOT

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: Various microcode, germ and boot files.

ETHERBOOT is a Envos Lisp background network server process which allows Dandelions and/or Doves (other than the one the server is running on) to boot utility programs from the Ethernet (as an alternative to floppies). On a Dandelion, a 3, 4 or 6 boot from the maintenance panel initiates an Etherboot; on a Dove the boot icons are used (sometimes in combination with a number key):

Dandelion	Dove	Boot Type
0003	F3	Ethernet non-diagnostic boot of the Installer
0004	F7	Ethernet diagnostic boot of the Installer
0006	F3-1	Ethernet boot of experimental software

(ETHERBOOT [LOGFILE]) [Function]

To start the server, (ADD.PROCESS '(ETHERBOOT)). LOGFILE is an optional argument which should be an open stream to log transactions in.

BOOTFILEDIRECTORIES [Variable]

The boot files are searched for on the directories in this list which should point to the (possibly remote) directory where the boot files are kept, initially '({CORE} {DSK}). The server will not respond to requests for boot files that are not available.

(CACHE.BOOT.FILES [TYPES]) [Function]

Since Lisp can take longer to open a remote file than the timeout on some (simple) requests, this function can be used to copy some of the boot files listed in ETHERBOOTFILES to the BOOTFILECACHEDIRECTORY. TYPES defaults to those listed in BOOTFILECACHETYPES.

BOOTFILECACHEDIRECTORY [Variable]

The directory into which CACHE.BOOT.FILES copies boot files, initially {CORE}.

BOOTFILECACHETYPES [Variable]

The default types of files that CACHE.BOOT.FILES copies to the BOOTFILECACHEDIRECTORY, initially '(DB GERM).

BOOTFILEREQUESTTYPES [Variable]

An association list which contains the type numbers of the requests that the boot server handles along with a description of the request type and the function which handles it. Currently, the request types are *Simple* and *SPP*.

## ETHERBOOTFILES

[Variable]

The table of boot file numbers and names. Each entry consists of a description of the boot file, the name of the file and the file number (48 bit) by which the file is requested. Since the boot server is table driven, different boot files can be substituted. Initially, ETHERBOOTFILES contains:

("Standard DLion Ethernet Initial Microcode"	EtherInitial.db	2852126720)
("Standard DLion Diagnostic Microcode"	MoonBoot.db	2852126728)
("Standard DLion Mesa Microcode"	Mesa.db	2852126736)
("Standard DLion Germ"	DLion.germ	2852126744)
("Standard DLion Boot File"	SimpleNetExecDLion.boot	2852126752)
("Standard DLion Diagnostics Boot File"	EIDiskDLion.boot	2852127232)
("Standard DLion Installer Boot File"	InstallerNSDLion.boot	2852127234)
("Alternate DLion Ethernet Initial Microcode"	EtherInitialAlt.db	2852126721)
("Alternate DLion Mesa Microcode"	Mesa.db	2852126738)
("Alternate DLion Germ"	DLion.germ	2852126746)
("Alternate DLion Boot File"	InstallerNSDLion.boot	2852126754)
("Standard TriDLion Diagnostic Microcode"	Moonboot.db	2852126729)
("Standard TriDLion Mesa Microcode"	TridentRavenMesa.db	2852126737)
("Standard TriDLion Germ"	TriDLion.germ	2852126745)
("Standard TriDLion Boot File"	SimpleNetExecTriDLion.boot	2852126753)
("Alternate TriDLion Mesa Microcode"	TridentRavenMesa.db	2852126739)
("Alternate TriDLion Germ"	TriDLion.germ	2852126747)
("Alternate TriDLion Boot File"	InstallerNSTriDLion.boot	2852126753)
("Standard Dove Ethernet Initial Microcode"	EtherInitialDove.db	2852128768)
("Standard Dove Diagnostic Microcode"	MoonRise.db	2852128776)
("Standard Dove Mesa Microcode"	MesaDove.db	2852128784)
("Standard Dove Germ"	Dove.germ	2852128792)
("Standard Dove Boot File"	SimpleNetExecDove.boot	2852128800)
("Alternate Dove Ethernet Initial Microcode"	EtherInitialDove.db	2852128769)
("Alternate Dove Diagnostic Microcode"	MoonRise.db	2852128777)
("Alternate Dove Mesa Microcode"	MesaDove.db	2852128785)
("Alternate Dove Germ"	Dove.germ	2852128793)
("Alternate Dove Boot File"	InstallerNSDove.boot	2852128801)
("Dove Simple Net Exec"	SimpleNetExecDove.boot	2852128824)
("Dove Configuration Utility"	SysConfigOfflineDove.boot	2852128825)
("Dove Installer"	InstallerNSDove.boot	2852128826)
("Dove Diagnostics Utility"	DiagDiskUtilDove.boot	2852128828)
("Dove Rigid Disk Diagnostics Utility"	DiagRDDove.boot	2852128829)
("Dove Ethernet Diagnostics Utility"	DiagEtherDove.boot	2852128830)
("Dove Keyboard & Display Diagnostics Utility"	KDMDove.boot	2852128831))

The boot file numbers overlay the host number space so Dandelion/Dove boot file numbers begin at 25200000000 octal.

## KNOWN PROBLEMS

- The server can only handle one connection at a time.
- Due to as yet unknown reasons, a Dandelion running the server is not able to service simple Dove requests; all other combinations should work.

---



---

## FILEWATCH

---



---

Johannes A. G. M. Koomen  
(Koomen.wbst@Xerox or Koomen@CS.Rochester)

This document last edited on October 19, 1987.

### INTRODUCTION

FILEWATCH is a facility for keeping an eye on open files. It periodically updates a display showing each open file stream, its current file pointer location, the total file size, a percentage bar, and a read/write/both indicator.

### DESCRIPTION

Invoking the function FILEWATCH (or selecting the "FileWatch" entry on the BackgroundMenu) starts up the FileWatch process if not already running, or brings up a FileWatch control menu allowing you to forget a currently displayed file (*i.e.*, stop displaying the file), recall a previously forgotten file, close an open file (after mouse confirmation), change some or all FileWatch display properties, or quit the FileWatch process. The Forget, Recall and Close entries on the FileWatch control menu have roll-outs to let you perform the operation on several files at once.

FileWatch can be customized by setting the FileWatch properties (see below) using the function FILEWATCHPROP. Right buttoning any FileWatch window brings up the FileWatch control menu, with the provision that the Forget and Close commands apply to the file displayed in that FileWatch window. Middle buttoning any FileWatch window allows you to move the entire FileWatch display, and left buttoning cause the window to be redisplayed.

### DETAILS

(FILEWATCH Command) [Function]

If Command is 'ON and no FileWatch process is already running, starts a process to watch open files. If Status is 'OFF or 'QUIT and there is a FileWatch process running, kills the process. If Command is neither one of the above nor one of the FileWatch commands listed below, starts a process to watch open files if not already running, otherwise brings up the FileWatch control menu. Returns the process if running, otherwise NIL.

FORGET [FileWatch command]

Brings up a menu of files currently being watched. Select the one you no longer want to have watched.

FORGET-MANY [FileWatch command]

Repeatedly performs the FORGET command until no other files are being watched or you make a null selection.

RECALL [FileWatch command]

Brings up a menu of forgotten files. Select the one you want to have watched again.



RECALL-MANY [FileWatch command]

Repeatedly performs the RECALL command until all forgotten files are being watched again or you make a null selection.

CLOSE [FileWatch command]

Brings up a menu of open files. Select the one you want to have closed.

CLOSE-MANY [FileWatch command]

Repeatedly performs the CLOSE command until all open files have been closed or you make a null selection.

MOVE [FileWatch command]

Performs the SET-ANCHOR, SET-POSITION, and SET-JUSTIFICATION commands.

SET-ANCHOR [FileWatch command]

Brings up a menu of four corner names. Select the one on you wish to anchor the FileWatch display. For instance, selecting Top-Right causes FileWatch windows to be stacked downwards with the top right corner of the first FileWatch window at the FileWatch display position.

SET-POSITION [FileWatch command]

Indicate where the FileWatch display should be positioned by moving the region of the combined FileWatch windows.

SET-JUSTIFICATION [FileWatch command]

Requests confirmation to turn FileWatch window justification on, *i.e.*, make all FileWatch windows the same width as the largest one.

(FILEWATCHPROP PropName [PropValue]) [Function]

If PropValue is given, sets the property value accordingly. Always returns the current (old) value of the property. This is a general facility which you can use for whatever purpose you deem appropriate. However, there are some properties that have a predefined meaning to FileWatch:

ALL-FILES? [FileWatch property]

If NIL, FileWatch displays only user visible open files; otherwise all open files (including, for example, dribble and file cacher files) are displayed. Initially set to NIL. Caveat: setting this property to T will give you access to things that might be dangerous to play with. In particular, closing certain system files on the Dorado may cause your machine to crash, and may leave the local file system in an unhealthy state.

ANCHOR [FileWatch property]

Each open file that is being watched gets its own FileWatch window. Multiple windows are stacked automatically. The total region occupied by this stack is anchored at the corner indicated by this property. The only legal values are TOP-LEFT, TOP-RIGHT, BOTTOM-LEFT, BOTTOM-RIGHT. Initially set to BOTTOM-RIGHT. If the anchor is at one of the bottom corners the stack grows upward, otherwise downward. If the anchor is at one of the left corners the stack is aligned by left edge, otherwise by right edge (see also the JUSTIFIED? property).

**FILTERS** [FileWatch property]

A list of file patterns, for example ("**{CORE}\*.\*;\***"). An open file that matches any of the patterns will not be watched. Initially set to NIL. Note that each pattern is expanded to include the **HOST** and **DIRECTORY** equal to that of (**DIRECTORYNAME**), **EXTENSION** and **VERSION** equal to "\*", unless already specified. For example, in my case, the filter "**\*JUNK\***" expands to "**{Ice}<Koomen>Lisp>\*JUNK\*.\*;\***". If you really wanted to filter all junk files, use the filter "**{\*}\*JUNK\***".

**FONT** [FileWatch property]

The font used for the FileWatch displays, specified in a form suitable to give to the function **FONTCREATE**. Initially set to '(GACHA 8).

**INTERVAL** [FileWatch property]

The value given to the function **BLOCK**. This should be either NIL or an integer indicating the number of milliseconds to wait between FileWatch display updates. Initially set to 1000. Note that FileWatch generates several **FIXP**'s for large files every time through the loop, so setting this to NIL may cause excessive storage allocation and reclamation.

**JUSTIFIED?** [FileWatch property]

If T all FileWatch windows are aligned along both left and right edges, and are grown or shrunk as needed to accommodate the maximum filename length currently in use. This is aesthetically more pleasing but incurs increased overhead due to frequent reshaping of the windows. Initially set to NIL.

**POSITION** [FileWatch property]

The location of the anchored corner of the FileWatch display. Initially set to the bottom right corner of the screen: (CONS SCREENWIDTH 0).

**SHADE** [FileWatch property]

The shade used for the FileWatch thermometers. Initially set to **GRAYSHADE**.

**SORTFN** [FileWatch property]

Either NIL or the name of a function taking two filenames as arguments (such as **ALPHORDER**), which is used to sort the list of open files being watched. Initially set to NIL (*i.e.*, no sorting).

---

---

**FILLREGION**

---

---

Originally By: Mike Bird (Inference Corp., Los Angeles, CA)

Jim Wogulis (Wogulis@ICS.UCI.EDU)

Greg Wexler (Wexler.pasa@Xerox)

New Owner: James M. Turner (Turner.Lexington@Xerox.com)

**INTRODUCTION**

The Fillregion package provides a function which will allow the user to "fill in" arbitrary regions of a bitmap or window with a shade or bitmap (or any valid shade argument to BITBLT).. The regions must be defined by a black or white outline. There are two functions provided to the user: FILL.REGION and AUTO.FILL.

(FILL.REGION *window.or.bm interior.pos shade*) [Function]

*window.or.bm* : Must be either a window or bitmap otherwise an error occurs.

*interior.pos* : Must be a position within window.or.bm that is within the interior of the region to be filled.

*shade* : Shade can be any valid shade argument that BITBLT will accept.

This will return the window.or.bm with the specified region filled in. The region to be filled is determined by the pixel specified at interior.pos. If the pixel is black, all the connected black regions will be shaded, otherwise, if the pixel is white, all the connected white region will be filled. If the user aborts the function before completion, the original window.or.bm will be restored.

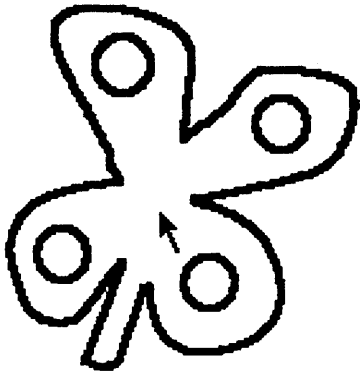
(AUTO.FILL *shade*) [Function]

*shade* : Shade can be any valid shade argument that BITBLT will accept.

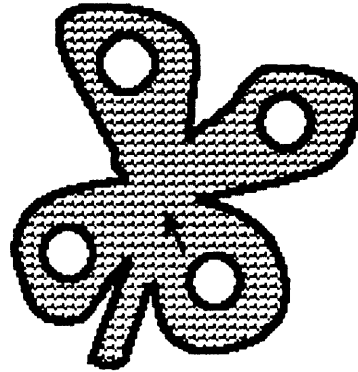
With your mouse pointing inside the appropriate region in a window, this function will fill in the region with the shade specified. This package only works for one bit per pixel bitmaps, color is not supported.

**Example:**

(AUTO.FILL 1234)



results in:



Comments and suggestions are welcome.

---

---

## FTPSEVER-MULTI- CONNECTIONS

---

---

By: Matt Heffron (BEC.HEFFRON@ECLA.USC.EDU)

Requires: FTPSEVER, FTPSEVERPATCH and DPUPFTPATCH

### INTRODUCTION

This package (actually a complimentary pair of files) extends the capabilities of the Lisp Library FTPSEVER package to support multiple simultaneous connections between Xerox 11xx series AI workstations.

### INSTALLATION

To install this package, load the FTPSEVERPATCH.LCOM file on the 11xx machine(s) that are to be servers (this will load FTPSEVER if it is not already loaded). Then load the DPUPFTPATCH.LCOM file on any of the 11xx machines that are to be clients of these servers. You must set the value of IL:\*FTP.NEGOTIATED.CONNECTION.HOSTS\* on each of the client machines to specify the server machines that support the FTPSEVERPATCH system of multiple simultaneous connections (below).

### VARIABLES

IL:\*FTP.NEGOTIATED.CONNECTION.HOSTS\* [Global Variable]

This variable must be set to specify the server machines that support the FTPSEVERPATCH system of multiple simultaneous connections. Its value is a list of PUP host numbers. (Specifically, it is a list of the values of (CAR (BESTPUPADDRESS <SERVER-HOST-NAME>)) for each of the server machines.)

### HOW IT WORKS

This package modifies the DPUPFTP code of the client machines, so that when it is trying to open an FTP connection BSP stream, it first checks to see if the server host is one of the IL:\*FTP.NEGOTIATED.CONNECTION.HOSTS\*, and if so, it sends a message to the modified FTPSEVER on that system (using PUP type \PT.NEGOTIATED.CONNECTION (= 128) on PUP socket \PUPSOCKET.NEGOTIATED.CONNECTION (= 63)). The server machine creates a socket for this connection and starts a standard FTPSEVER listener process on this socket, and returns the socket number to the client. (The process is modified so it will go away when the connection is closed instead of lingering forever.) The client uses the returned socket number for the connection instead of \PUPSOCKET.FTP. If the server is NOT on IL:\*FTP.NEGOTIATED.CONNECTION.HOSTS\*, or fails to respond within 10 seconds with the new socket number, then \PUPSOCKET.FTP is used. When the negotiated connection server is started on the server machine (with the incantation (FTPSEVER) which is the original FTPSEVER start up), it also will start up a *permanent* FTPSEVER listener on \PUPSOCKET.FTP so regular connection requests can be handled.

### ACKNOWLEDGEMENTS

Thanks to Tom Lipkis of Savoir for suggesting this sort of scheme.

---



---

## GRAPHCALLS

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: GRAPHER, MSANALYZE (WHERE-IS & HELPSYS optional)

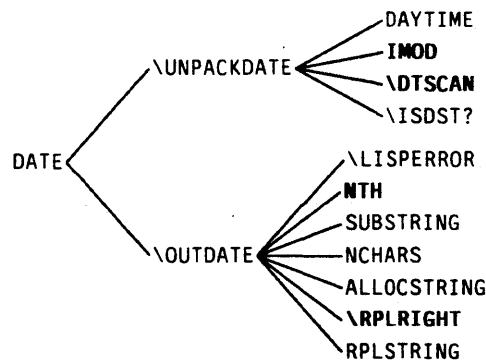
GRAPHCALLS is an extended graphical interface to the Envos Lisp CALLS function. It is to CALLS what BROWSER is to SHOW PATHS in MASTERSCOPE. It allows fast graphing of the calling hierarchy of both interpreted and compiled code, whether or not the source is available (see the CALLS function in the MASTERSCOPE section of the *Lisp Library Modules* manual), allowing examination of both user and system functions. The sources of the functions do not have to be analyzed by MASTERSCOPE first.

Additionally, buttoning a function on the graph brings up a menu of operations that can be done with the function, such as editing, inspecting, further graphing etc.

(GRAPHCALLS *FUNCTION* &REST *OPTIONS*)

[Function]

Graphs the calling hierarchy of *FUNCTION*. Terminal nodes on the graph (those which call no other functions or are undefined) are printed in a bold version of the graph's font indicating that they cannot be graphed further:



The remainder of the arguments, in keyword format, make up *OPTIONS* eg.

```
(GRAPHCALLS 'DATE :FONT '(GACHA 10) :DEPTH 4 :FILTER 'FGETD)
```

Options include:

- :STREAM**      An image stream to display the graph on. The options list is saved on the stream.
- :FILTER**      A predicate to apply to the functions when building the graph to test their eligibility to appear on the graph. The filter can be any defined function; the default is not to filter. Interesting filters include:
  - WHEREIS**      Limits the tree to only functions the user as has loaded and prunes out system functions and SYSLOADED files. Quite useful.

- FGETD** Limits the tree to only functions that are actually defined. Thus if you are perusing the tree for BITBLT and do not have and are not interested in the color code, FGETD will remove all of the undefined color bitmap functions.
- EXPRP** Limits the tree to interpreted functions. Useful for graphing functions in the development stage.
- CCODEP** Limits the tree to compiled functions.
- NO\** Keeps low level functions starting with \ (i.e. \OUTDATE) off of the graph. Useful for getting an overview of system functions and when advising system functions (as \ed functions should probably not be advised).
- :DEPTH** The calling hierarchy is graphed to *depth* levels (defaults to 2).
- :FORMAT** Passed to LAYOUTGRAPH and can be any format specification (LATTICE, VERTICAL, REVERSE etc.); defaults to (HORIZONTAL COMPACT REVERSE/DAUGHTERS). In the forest format multiple instances of a function appear on the graph after every calling function and a boxed node indicates the function appears elsewhere on the graph, possibly graphed further. In the lattice format each function gets placed on the graph only once (particularly useful for dynamic graphing, described below), and boxed nodes indicate recursive functions calls.
- :SEARCHFN** A function to use to generate the children of a given node. It should return a list whose first item is a list of the children, the other items in the list are ignore. Using this feature, it is possible to graph things other than functions. To graph what files load other files, supply a search function of (LAMBDA (FILE) (LIST (FILECOMSLST FILE 'FILES))) and a file name for the function argument.
- :ADVISE** Advises the functions after they are graphed (see *Dynamic Graphing* below); recognized values are one or both of the following:
- INVERT** Visually tracks a running program .
- COUNT** Counts function calls in a running program.
- :DELAY** The delay to use in advised graphs; defaults to 500 milliseconds.
- :NAMEFN** A function to use to generate the node labels on the graph.
- :FONT** The font to use to display the graph; defaults to (GACHA 8).
- :SHAPE** A boolean that indicates if the window should be shaped to fit the graph; defaults to NIL.
- :PRIN2FLG** A boolean that indicates to use PRIN2 when printing node labels, defaults to NIL.
- :SUBFNDEFFLG** A boolean that enables graphing of compiler generated functions; defaults to T.
- :TOPJUSTIFYFLG** Passed to SHOWGRAPH; defaults to NIL.
- :ALLOWEDITFLG** Passed to SHOWGRAPH; defaults to NIL.

**GRAPH MENUS**

The menu that pops up when you left button a function on the graph contains the following items:

<b>? =</b>	Print the arguments to the function, if available.										
<b>HELP</b>	Calls HELPSYS on the function.										
<b>FNTYP</b>	Print the function's FNTYP.										
<b>WHERE</b>	Do a WHEREIS (with FILES = T) on the function.										
<b>EDIT</b>	Calls the editor on the function if available for editing.										
<b>TYPEIN</b>	BKSYSBUFs the name of the function into the typein buffer.										
<b>BREAK</b>	Applies BREAK to the function. Its subitems are: <table> <tr> <td><b>BREAKIN</b></td> <td>Breaks the function only in the context of a particular calling function. In lattice format, if the function has more than one function calling it on the graph, the user is prompted to indicate the caller in which to break the function.</td> </tr> <tr> <td><b>UNBREAKIN</b></td> <td>Undoes BREAKIN.</td> </tr> <tr> <td><b>UNBREAK</b></td> <td>Applies UNBREAK to the function.</td> </tr> <tr> <td><b>TRACE</b></td> <td>Applies TRACE to the function.</td> </tr> <tr> <td><b>TRACEIN</b></td> <td>Traces the function only when called from inside a particular function, like BREAKIN above. Use UNBREAKIN to remove the trace, or else UNBREAK on the window menu.</td> </tr> </table>	<b>BREAKIN</b>	Breaks the function only in the context of a particular calling function. In lattice format, if the function has more than one function calling it on the graph, the user is prompted to indicate the caller in which to break the function.	<b>UNBREAKIN</b>	Undoes BREAKIN.	<b>UNBREAK</b>	Applies UNBREAK to the function.	<b>TRACE</b>	Applies TRACE to the function.	<b>TRACEIN</b>	Traces the function only when called from inside a particular function, like BREAKIN above. Use UNBREAKIN to remove the trace, or else UNBREAK on the window menu.
<b>BREAKIN</b>	Breaks the function only in the context of a particular calling function. In lattice format, if the function has more than one function calling it on the graph, the user is prompted to indicate the caller in which to break the function.										
<b>UNBREAKIN</b>	Undoes BREAKIN.										
<b>UNBREAK</b>	Applies UNBREAK to the function.										
<b>TRACE</b>	Applies TRACE to the function.										
<b>TRACEIN</b>	Traces the function only when called from inside a particular function, like BREAKIN above. Use UNBREAKIN to remove the trace, or else UNBREAK on the window menu.										
<b>CCODE</b>	Calls INSPECTCODE on the function if it is compiled code.										
<b>GRAPH</b>	Calls GRAPHCALLS to make a new graph starting with function, inherits the original graph's options.										
<b>FRAME</b>	Inspect the local, free and global variables of the function. These are the last three lists of the CALLS function placed into INSPECT windows. Its subitems are: <table> <tr> <td><b>&gt;FRAME</b></td> <td>Like FRAME but for all of the functions on the sub-tree starting at the selected node and only for FREEVARS and GLOBALVARS.</td> </tr> <tr> <td><b>&lt;FRAME</b></td> <td>Like &gt;FRAME but for all of the functions above the function in the graph, i.e. the FREEVARS and GLOBALVARS in the function's scope.</td> </tr> </table>	<b>&gt;FRAME</b>	Like FRAME but for all of the functions on the sub-tree starting at the selected node and only for FREEVARS and GLOBALVARS.	<b>&lt;FRAME</b>	Like >FRAME but for all of the functions above the function in the graph, i.e. the FREEVARS and GLOBALVARS in the function's scope.						
<b>&gt;FRAME</b>	Like FRAME but for all of the functions on the sub-tree starting at the selected node and only for FREEVARS and GLOBALVARS.										
<b>&lt;FRAME</b>	Like >FRAME but for all of the functions above the function in the graph, i.e. the FREEVARS and GLOBALVARS in the function's scope.										

Buttoning the graph outside a node give you a menu with these options:

<b>UNBREAK</b>	Does an (UNBREAK), unbreaking all broken functions.
<b>RESET</b>	Resets the counters for the COUNT option and redisplay the graph.

#### DYNAMIC GRAPHING

When the ADVISE option is specified with the value(s) of INVERT and/or COUNT, GRAPHCALLS will advise all of the functions on the graph (in the context of their parent) to invert their corresponding node on the graph (as well as delay some to allow it to be seen) and/or follow each function name by a count of the number of times it has been executed. In invert mode, a node remains inverted as long as control is inside its corresponding function and it returns to normal



when the function is exited. The lattice format is best when using the invert feature. Closing the graph window UNADVISEs the functions on the graph.

An example of this is (GRAPHCALLS 'DATE :ADVISE 'INVERT) and then evaluate (DATE).

GRAPHCALLS will not graph or advise any function in the system list UNSAFE.TO.MODIFY.FNS when the advise option is used. Functions which are unsafe to advise should be added to this list.

**CAVEAT PROGRAMMER!** This feature must be used with caution. As a rule, one should not do this to system functions, but only one's own, use WHEREIS as a filter for this. Advising system code indiscriminately will probably crash the machine unrecoverably.

You can, at some risk, interactively break and edit functions on the graph while the code is executing. Also, creating subgraphs of advised graphs will show the generated advice functions not the original functions called, as will creating new graphs of functions in advised graphs. You can create advised graphs of functions already graphed normally on the screen.

### COMMAND WINDOW

GraphCalls Command Window					
Command	Filters	Flags	Format	Depth	Delay
<b>Function</b>	WhereIs	Invert	Lattice	0	0
Include	FGetD	Count	Reverse	1	1
Exclude	ExprP	Shape	Vertical	2	2
Clear	CCodeP	Edit	ArgList	3	3
<b>Graph</b>	No\	Prin2	WhereIs	4	4
				5	5
				6	6
				7	7
				8	8
				9	9
				10	10

(GRAPHCALLSW [REBUILD?])

[Function]

Puts up a command window with menus that will interactively set up calls to GRAPHCALLS. The menus let you set the Invert, Count and Edit flags, select from common filters and formats and set the depth of the graph. You can also change the amount of delay used in the advised functions when doing dynamic graphing. If you specify an advised graph (Invert or Count) and do not specify a WHEREIS filter, you will be asked to confirm with the mouse for your own protection.

More than one item on the filter and flags menus can be selected at a time. Buttoning a selected item on these menus unselects it. The command menu contains the following:

- Function**      Prompts for the name of a function to graph when the **Graph** item is selected.
- Include**       Adds files or functions to the list of items to allow on the graph, see the Include/Exclude algorithm below.
- Exclude**      Adds files of functions to the list of items disallowed on the graph, see the Include/Exclude algorithm below.

**Clear** Clears all of the settings on the command window to their defaults. Also clears the Include/Exclude lists.

**Graph** Graphs the function by calling GRAPHCALLS with the selected options.

Include and Exclude allow fine tuning of the filter function. If the function passes the filter, then the following are tried until one determines whether or not the function will be on the graph:

If a set of functions has been explicitly excluded, and the function is a member of this set, it will NOT appear on the graph.

If a set of functions has been explicitly included, and the function is a member of this set, it WILL appear on the graph.

If a set of files has been explicitly excluded, and the function is in one of those files, it will NOT appear on the graph.

If a set of files has been explicitly included, and the function is not in one of those files, it will NOT appear on the graph.

The function WILL appear on the graph.

The format menu contains two items that are not passed on to GRAPHER but are used to select alternate NAMEFN options:

**ArgList** Supplies a NAMEFN that will print the function and its arguments (using SMARTARGLIST) as the node label.

**WhereIs** Supplies a NAMEFN that will print the function followed by the file(s) found by doing WHEREIS (with FILES = T) if any .

When the command window is open, middle buttoning a node on a GRAPHCALLS graph will bring up a menu of commands relating to command window and graphs. The menu contains:

**EXCLUDE** Adds the function to the exclude functions list of the command window. This is the only way to exclude system functions which get added to the SYSTEM file exclusion list.

The command window can also be obtained via the background menu. Subsequent calls to GRAPHCALLSW (either directly or via the background menu) will reuse the old command window if there is one. If this window is damaged, and redisplay does not help, then setting *REBUILD?* to T will build a new command window from scratch.

## NOTES

- Function call graphs are constructed using breadth first search but GRAPHER lays out graphs depth first so functions may be expanded in different places on the graph than expected.
- GRAPHCALLS sysloads GRAPHER and MSANALYZE if they are not already loaded.
- In dynamic graphs, variables caused by advising show up in the frame inspections.
- The global variable GRAPHCALLS.DEFAULT.OPTIONS contains all of the defaults for GRAPHCALLS keywords, in property list format.

---

---

**GREP**

---

---

By: Larry Masinter (Masinter.pa@Xerox.com)

Requires: BSEARCH

**INTRODUCTION**

like FGREP of Unix: searches for strings in files.

(GREP *STRS FILES*) [Function]

STRS is a string or a list of strings. FILES is a file or a list of files. Searches for the given string(s) in the given file(s), showing each line.

(PHONE *name*) [Function]

Calls (GREP name PHONELISTFILES). PHONELISTFILES is initialized to NIL. (The PARC init file resets it to point to the PARC phone list.)

For example,

(GREP (QUOTE ED) (QUOTE {INDIGO}<REGISTRAR>PARCPHONELIST.TXT))

will print:

```
(from {INDIGO}<REGISTRAR>PARCPHONELIST.TXT;3)
4183 <Endicott>, Fred 35-1354
4435 <Fiala>, Ed 35-2166
4598 <RKennedy>, Ray 34-78
4839 <McCreight>, Ed 35-2146
5759 Pedersen, Jan 32-202
4818 Satterthwaite, Ed * 35-2174
MES Solcz, Edward J. 8* 348-1214
ATA Wahlenmeier, Fred 887-4018
```

**GRID-ICONS**

By: sML (Lanning.pa@Xerox.com)

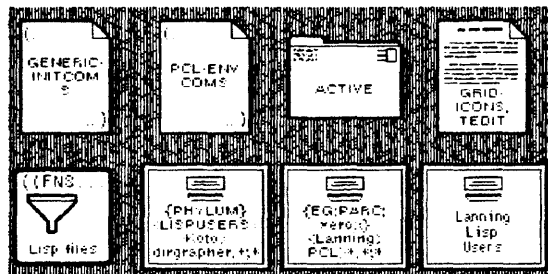
Last edited: September 14, 1987

**INTRODUCTION**

Grid-Icons provides the Lisp user with a set of default window icons that resemble those found in the Viewpoint system. There is an option that the user can set to force these icons to be positioned on a grid, instead of the unrestricted positioning allowed by Lisp.

**USING GRID-ICONS**

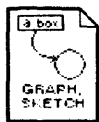
All that is required is loading the file GRID-ICONS. When the file is loaded, it redefines a number of standard window icons in the system.



**REDEFINED ICONS**



[TEdit icon]

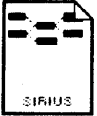


[Sketch icon]

Note that GRID-ICONS not only redefines TEdit and Sketch icons, it also changes the way that the icon title is computed: the host and directory information is removed, so that only the name and extension remain.



[SEdit icon]

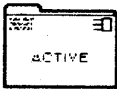


[Loops Browser icon]



[Default icon]

In Lyric, it is possible to redefine the standard icon used by the system. GRID-ICONS uses this, and redefines the standard system icon in LYRIC.



[Lafite mail folder icon]

**NEW ICONS**

GRID-ICONS defines a handy icon for accessing the list of files that have been loaded into you system.



[Lisp files icon]

Buttoning on this icon will pop up a menu of all loaded files (as determined by the value of the variable FILELST). Selecting a file from this menu will open up an editor on the COMS of that file. There is an additional item on the menu, "\* New file \*", that can be used create a new file, and then edit its COMS. This icon window is stored in the variable LOADED-FILES-ICON-WINDOW.

**USER FUNCTIONS**

The user can declare that any given window stick to grid positions.

(GRID-WINDOW window)

[Function]

Causes window to pay attention to ENFORCE.ICON.GRID; if the value of ENFORCE.ICON.GRID is true, the window will make sure that it is centered on a grid location. By default the spy button and icons produced by the ICONW and TITLEICONW functions pay attention to ENFORCE.ICON.GRID.

**VARIABLES THAT CONTROL GRID-ICONS**

There are a few variables that control how GRID-ICONS works.

**ENFORCE.ICON.GRID** [Variable]

If ENFORCE.ICON.GRID is true, window icons (and any window declared "gridded" by the GRID-WINDOW function) will be restricted to be positioned on a grid. The default value of ENFORCE.ICON.GRID is NIL.

**IGNORE.ICON.GRID** [Window property]

The IGNORE.ICON.GRID window property provides a way to control icon gridding on an icon-by-icon basis. If the IGNORE.ICON.GRID window property of an icon is true, the icon will not be restricted to grid positions. This window property is checked only if ENFORCE.ICON.GRID is true.

**ENFORCE.ICON.REGIONS** [Variable]

You can enforce icon gridding in individual regions of the screen by using the variable ENFORCE.ICON.REGIONS. If the value of ENFORCE.ICON.GRID is true, and the icon does not have a IGNORE.ICON.GRID window property, the proposed new position for the icon is tested against the value of ENFORCE.ICON.REGIONS. If ENFORCE.ICON.REGION is NIL, gridding is enforced as described above. Otherwise, ENFORCE.ICON.REGION should be a list of regions; gridding will be enforced only if the proposed position is within one of these regions. The default value of ENFORCE.ICON.REGIONS is NIL. [Thanks/blame to Ramana Rao for this.]

**ICON.SIZE** [Variable]

ICON.SIZE specifies the maximum size of the icons, for use in computing the grid positions of icons. It is a cons of the maximum width and the maximum height. The default value is (85 . 85), which is the "correct" value for the icons defined in this utility.

**ICON.SPACING** [Variable]

ICON.SPACING specifies the gap between icons, for use in computing the grid positions of icons. It is a cons of the horizontal gap and the vertical gap. The default value is (5 . 5).

**GRID.OFFSET** [Variable]

GRID.OFFSET specifies origin of the icon grid. The default value is (0 . 0).

**DEFAULTICONFONT** [Variable]

The value of DEFAULTICONFONT is the default font used by the system when printing titles in icons. Since the icons defined in GRID-ICONS tend to be smaller than the original icons, you might want to use a slightly smaller font than the default. Personally, I recommend setting DEFAULTICONFONT to (FONTCREATE '(HELVETICA 8)).

---

---

## Hanoi

---

---

By: Larry Masinter (Masinter.pa@Xerox.com)

### INTRODUCTION

Ancient graphics demo, upgraded to be idle hack. Adds Hanoi to list of idle displays.

### OPERATION

(HANOI *NRINGS WINDOW FONT ONCE*)

[Function]

Will display in *WINDOW* (or *HANOIWINDOW*, created first time) a towers-of-hanoi problem and solve it. It periodically blocks so you can run it as a background process. *NRINGS* is the number of rings. If *NRINGS* is a list it is the labels printed on the rings in font *FONT*. It conforms to the window shape if you reshape it. It will run indefinitely unless *ONCE* is non-NIL.

---



---

## HASHBUFFER

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: HASH

HASHBUFFER combines *hash files* with *hash arrays* in order to improve hash file performance when keys are accessed multiple times. This module also defines two functions for moving data between hash files and hash arrays.

The functions below are used in place of the hash file routines. When a hash file is opened, a hash array is created, of a complimentary size. When requests for keys are made, the array is searched, and if a value is found, it is returned. If a value is not found, the file is searched and if a value is found there, it is stored in the array and returned. If a value is not found, a marker is put in the array so that the file is not searched again.

(OPENHASHBUFFER *FILE [ACCESS MINKEYS OVERFLOW HASHBITSFN EQUIVFN]*) [Function]

Opens an existing hash file and returns a hash buffer datum which must be given to the other hash buffer functions. Only the *FILE* argument is required; the *MINKEYS* argument is used for the size of the hash array and if not supplied the size of the hash file is used. Setting *MINKEYS* smaller than the size of the hash file allows a fast, small hash array window onto a larger, slower hash file. The *OVERFLOW*, *HASHBITSFN* and *EQUIVFN* arguments are passed to HASHARRAY.

(CREATEHASHBUFFER *FILE [VALUETYPE ITEMLength #ENTRIES`  
OVERFLOW HASHBITSFN EQUIVFN]*) [Function]

Like OPENHASHBUFFER but creates a new hash file. The *FILE*, *VALUETYPE* and *ITEMLength* arguments are passed to CREATEHASHFILE; the *OVERFLOW*, *HASHBITSFN* and *EQUIVFN* arguments are passed to HASHARRAY. The *#ENTRIES* argument is used for both the file and array.

(CLOSEHASHBUFFER *HASHBUFFER [FILEONLY?]*) [Function]

Closes the hash file and sets the hash array to NIL so that it can be reclaimed. If *FILEONLY?* is non-NIL then only the hash file is closed, the hash array will be left alone.

(GETHASHBUFFER *KEY HASHBUFFER*) [Function]

(PUTHASHBUFFER *KEY VALUE HASHBUFFER*) [Function]

Retrieve and store *VALUE* for *KEY* in the hash buffer. If the hash file is only open for input, then storing a key will only affect the hash array. If the hash file is open for output, then storing a key will put it in both the hash array and hash file. If *VALUE* is NIL, then a delete is performed.

(HASHARRAY.TO.HASHFILE *HASHARRAY HASHFILE [TESTFN]*) [Function]

Uses MAPHASH to move the contents of *HASHARRAY* into a hash file. If *HASHFILE* is a file name, CREATEHASHFILE is called; if *HASHFILE* is an open hash file datum, it is used and left open. *TESTFN*, if supplied, is called before each PUTHASHFILE on (KEY VALUE HASHARRAY HASHFILE) and if it returns non-NIL, the key and value are copied to the file.



(HASHFILE.TO.HASHARRAY *HASHFILE* [*HASHARRAY TESTFN*])

[Function]

Uses *MAPHASHFILE* to move the contents of *HASHFILE* into a hash array. If *HASHARRAY* is not supplied a new hash array is created. *TESTFN* is called before each *PUTHASH* on (KEY VALUE *HASHFILE* *HASHARRAY*) and if it returns non-NIL, the key and value are copied to the array.

---



---

## HASHDATUM

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: HASH

HASHDATUM facilitates storing random Envos Lisp datatypes on *hash files* using the hashed text feature of the HASH Lisp Library module. The module defines functions which access an item on a hash file as a stream of bytes using user supplied input and output functions. Since the items are stored using text hashing, when rehashing or copying of the file occurs, the data portion of the file is copied correctly.

(GETHASHDATUM KEY HASHFILE READFN) [Function]

(PUTHASHDATUM KEY DATUM HASHFILE PRINTFN) [Function]

Use *READFN* and *PRINTFN* to store and retrieve *DATUM* on *HASHFILE*. The *READFN* takes a stream as its argument, the *PRINTFN* takes the *DATUM* and a stream. The *put* function returns the hash file text pointer record which contains two byte pointers that indicate where the datum begins and ends on the file. The *get* function returns the result of the *READFN*.

The following macros and functions are also defined using the above functions:

(GETHASHGRAPH KEY HASHFILE) [Macro]

(PUTHASHGRAPH KEY GRAPH HASHFILE) [Macro]

Use *GRAPHER* functions *READGRAPH* and *DUMPGRAPH* to store *GRAPH* on *HASHFILE* under *KEY*.

(GETHASHBITMAP KEY HASHFILE) [Macro]

(PUTHASHBITMAP KEY BITMAP HASHFILE) [Macro]

Use *READBITMAP* and *PRINTBITMAP* to store *BITMAP* on *HASHFILE* in a text format.

(GETHASHBINARYBITMAP KEY HASHFILE) [Macro]

(PUTHASHBINARYBITMAP KEY BITMAP HASHFILE) [Macro]

Use *READBM* and *WRITEBM* from *BITMAPFNS* to store *BITMAP* on *HASHFILE* in a binary format.

(GETHASHTEDIT KEY HASHFILE [WINDOW PROPS]) [Function]

(PUTHASHTEDIT KEY TEXTOBJ HASHFILE) [Macro]

Use *OPENTEXTSTREAM* and *TEDIT.PUT.PCTB* from *TEDIT* to store *TEXTOBJ* on *HASHFILE*, preserving both the text and formatting information. *WINDOW* and *PROPS* are optional and are passed to *OPENTEXTSTREAM*. If the *WINDOW* argument is not supplied, the result of the *get* function can be passed to *OPENTEXTSTREAM* along with a window to display the text.

(GETHASHUGLY KEY HASHFILE) [Macro]

(PUTHASHUGLY KEY UGLYVAR HASHFILE) [Macro]

Use *HREAD* and *HPRINT* to store random data, like menus, on *HASHFILE*.

---



---

## HEADLINE

---



---

By: D. Austin Henderson, Jr. (AHenderson.pa@Xerox.com)

Last revised: April 1, 1986

**HEADLINE** contains functions for creating and closing windows which contain headlines ("headline windows").

**(HEADLINE PHRASE FONT POSITION ALIGNMENT)** [Function]

Creates a headline window with *PHRASE* printed in font *FONT* at position *POSITION* aligned as per *ALIGNMENT*; the window is just large enough to hold the headline. *PHRASE* is any Lisp object. *FONT* defines a font as per FONTCREATE (eg. (TIMESROMAN 18 BOLD) ); if NIL, TimesromanD 36 is used. *POSITION* is a position giving the reference point for placing the window; if NIL, the user is given a chance to position the window with MOVEW. If *POSITION* is given, *ALIGNMENT* gives the alignment of the window with respect to *POSITION* as (xalignment . yalignment) where xalignment is one of LEFT, CENTER, or RIGHT and yalignment is one of BOTTOM, CENTER, or TOP; for convenience, if Position is CENTER then it is taken to mean (CENTER . CENTER), etc.

**(HEADLINE.ARRAY TITLES ALIGNMENT SEPARATION POSITION)** [Function]

Creates a set of vertically arranged headline windows. *TITLES* is a list of (phrase font) sublists where phrase and font are as in Headline. *ALIGNMENT* is one of LEFT, CENTER, or RIGHT, indicating how the windows are aligned with each other; defaults to CENTER. *SEPARATION* indicates the spacing between the bottoms of the windows; defaults to 70. *POSITION* indicates where the top (first) of the windows is to appear; defaults to somewhere near the top center of the screen.

**(BILLBOARD)** [Function]

Identical to HEADLINE.ARRAY, left in for backward compatibility.

**(BANNER PHRASE FONT POSITION ALIGNMENT)** [Function]

Same as HEADLINE except it prints the phrase vertically.

**(BANNER.ARRAY TITLES ALIGNMENT SEPARATION POSITION)** [Function]

Same as HEADLINE.ARRAY except it prints the phrases vertically, left to right.

**(CLOSE.HEADLINES)** [Function]

Closes all the active headline windows.

---



---

## HPGL

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

HPGL defines a Lisp *image stream* type that generates output for plotters (and other devices) which use the *Hewlett-Packard Graphics Language*. The module does not define any user functions, the HPGL streams are accessed via OPENIMAGESTREAM and the hardcopy functions.

### PLOTTERS

Some plotters which use HPGL either as their primary language or as an optional extra:

Hewlett-Packard (most) Facit 4551 Gould Color-writer 6120 and 6320 Taxan 710 IBM 7371 Roland DYX-880 and 980	Epson America HI-80 (option) Western Graphtec MP1000 and FP5301 (option) Houston Instrument DMP-29 (option) Nicolet Zeta 8 (option)
---	--

(source: PC, Volume 4, Number 26, December 1984)

The file extensions HPGL and PLOT are recognized by the system as plotter output file types.

### OPTIONS

The driver accepts the following in the OPTIONS argument to OPENIMAGESTREAM:

<b>SCALE</b>	Image scaling; value should be a POSITION record which indicates where the second scaling point should be placed (the initial scaling point is at 0,0). By default, uses ( SCREENWIDTH . SCREENHEIGHT ).
<b>ROTATE</b>	Paper rotation; value should be 0 or 90, defaults to <i>landscape</i> plotting (0).
<b>PAPER</b>	Paper size; value is a small integer, the HP 7475A accepts 3 (A3) or 4 (A4).
<b>TERMINATOR</b>	Label terminator character; value should be a character, the default is '↑A'.
<b>VELOCITY</b>	Pen velocity; plotter specific.

### IMPLEMENTATION

The driver was implemented using an HP 7475A plotter but the plotter output conforms to the more restrictive HP 9872 syntax to be more widely applicable. The driver uses the following variables which may need adjustment for other types of plotters:

<b>HPGL.FONTS</b>	An ALST of font names and (small integer) plotter font numbers.
<b>HPGL.OPTIONS</b>	An ALST of plotter specific options that can be passed to OPENIMAGESTREAM and the corresponding HPGL command to print.
<b>HPGL.DASHING</b>	An ALST of HPGL line types (small integers) and dashing lists.
<b>HPGL.FONT.EXPANSIONS</b>	An ALST of font face expansions (REGULAR, COMPRESSED and EXPANDED) and the relative scale of each.
<b>HPGL.TERMINATOR</b>	The default end of instruction terminator character, initially ';'.

HPGL.SEPARATOR	The default parameter separator character, initially ','.
HPGL.TEXT.TERMINATOR	The default end of label terminator character, initially '↑A'.
HPGL.CHORD.ANGLE	The chord angle used by the circle and arc instructions. Defaults to NIL which causes the plotter's default to be used.
HPGL.PATTERN.LENGTH	The default pattern length for the hardware dashing. Defaults to NIL which causes the plotter's default to be used.
COLORNAMES	System variable used to convert between RGB triples and pen numbers. The order of entries affects the pen number to color correspondences.

**DASHING**

To minimize the complexity of the driver and maximize the speed of plotting, for operations other than DRAWLINE, the driver only uses the built-in dashing types of the plotter. The correspondences between the dashing style and HPGL line type number are kept in the HPGL.DASHING variable which can be modified or extended for plotters with different dashing styles than those displayed below:

1	.....	( 1 49)
2	_____	( 25)
3	_____	( 35 15)
4	_____	( 39 5 1 5)
5	_____	( 35 5 5 5)
6	_____	( 25 5 5 5 5 5)
	_____	NIL

If the driver is loaded after SKETCH, the dashing types are added to SKETCH's dashing menu .

---



---

## IDLEHAX

---



---

By: Larry Masinter (Masinter.pa@Xerox.com) with contributions by various others.

### INTRODUCTION

This module contains a couple of random demonstration programs, useful as "Idle programs", callable from the background menu. The Idle display options includes Lines Warp-Out Radar Triangles RandAngles Polygons Bubbles and Kaleidoscope.

These are implemented by the following functions:

**(POLYGONS *W NOBLOCK TIMER*)** [Function]

Calls (POLYGONS) or (POLYGONS window) to perpetually draw polygons in the given window (it (re)uses POLYGONSWINDOW if argument is NIL). To run in the background, you can ADD.PROCESS((POLYGONS (CREATEW)). Controlled somewhat by the global parameters POLYGONMINPTS (minimum number of vertices), POLYGONMAXPTS (maximum number of vertices), POLYGONSTEPS (number of steps between min and max), and delays POLYGONWAIT (time between different polygons) and POLYGONWAIT2 (delay between initial display of beginning and end and the movement phase.)

If NOBLOCK is T, it doesn't block at all (runs after but can't run in background.) If TIMER is given, then POLYGONS will stop after TIMER is expired. (Used by the demo system.)

**(LINES *W N LCNT STEPS ODDSTEP*)** [Function]

Similar to POLYGONS in controls, but draws perpetually changing form using line draw. W defaults a "demo window", but is the window on which the display is drawn, N is the number of endpoints (e.g., 2 draws lines, 3 draws triangles, 7 draws 7-segment figures), LCNT is the "number of lines on the screen at any one time", STEPS is the number of lines to draw between start and end (the higher this number, the closer together the lines are), and ODDSTEP is a flag: if T, then the odd endpoints remain the same every other iteration (try (LINES NIL 3 1 40 T).) The background RandAngles means: (LINES W (RAND 3 7) (RAND 1 16) (RAND 25 100)), while Triangles is (LAMBDA (W) (LINES W 3 1 40)), etc.

**(BUBBLES *WINDOW*)** [Function]

Perpetually draws circles. Controlled by BUBBLECNT, which is read at startup as the number of circles visible at any one time.

**(KAL *W PERIOD PERSISTENCE*)** [Function]

Borrowed from the KAL LispUsers package: draws a random symmetric pattern of dots. Pretty. Period affects the style of display, while PERSISTENCE affects how many dots are on the screen at once.

**(WARP *W*)** [Function]

Draws a sequence of circular patterns that resemble piles of sand. Or not; you decide.

**POLYGONS, LINES and BUBBLES** adjust themselves to the size of the window, so you can reshape the window in the middle of the demo.

---



---

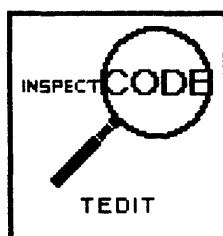
**INSPECTCODE-TEDIT**


---



---

By: Matt Heffron (BEC.HEFFRON@ECLA.USC.EDU)  
 Beckman Instruments, 2500 Harbor X-11  
 Fullerton, CA. 92634  
 (714) 961-3128



The **INSPECTCODE-TEDIT** package advises the **INSPECTCODE** facility to have some extended capabilities when the **TEDIT** and **GRAPHCALLS** packages are loaded (i.e. it uses **TEDIT** and **GRAPHCALLS**).

If **TEDIT** is not defined, then the standard **INSPECTCODE** will be used. If **TEDIT** is loaded, then a read-only **TEDIT/INSPECTCODE** window will be opened, and will have a special **INSPECTCODE** menu for **LEFT** or **MIDDLE** buttoning in the titlebar. All of the options, except for **Quit**, in this menu use the current selection in the window. You make selections with the mouse buttons in the standard **TEDIT** ways. The options in the **INSPECTCODE** titlebar menu are:

- |                           |  |
|---------------------------|--|
| <b>GraphCalls</b>         | If the <b>GRAPHCALLS</b> package is loaded, then calls <b>GRAPHCALLS</b> on the current selection.   |
| <b>InspectCode</b>        | Opens a new <b>INSPECTCODE</b> window on the current selection.  |
| <b>Inspect</b>            | Does an <b>INSPECT</b> on the value of the current selection. This item has <b>SUBITEMS</b> (see below).   |
| <b>Pretty Print Value</b> | Prompts for region to open a window, and prettyprints the value of the current selection in it. This item has <b>SUBITEMS</b> (see below).                                 |
| <b>Quit</b>               | Closes this window and kills the associated <b>TEDIT</b> process. (Closing the window with the <b>WindowMenu</b> , or by calling <b>CLOSEW</b> on it does the same thing.) |

The **Inspect** and **Pretty Print Value** menu options have the following **SUBITEMS** which affect how the value of the current selection is determined:

- |               |   |
|---------------|---|
| <b>Freely</b> | The value of the current selection is determined by any binding that a free-reference from the <b>INSPECTCODE</b> window menu handling code (i.e by |
|---------------|---|



(EVALV selection)). This is the default behavior when a menu selection is made directly from the titlebar menu without using the SUBITEMS menu.

**Globally**

The value of the current selection is determined by its top level (Global) binding.

**In Process Context**

The value of the current selection is determined by its binding in the context of a specified process. A menu of all current processes will be brought up to allow you to specify a process.

INSPECTCODE-TEDIT also defines the LISPXMATCH IC which INSPECTCODE's its argument.

---

---

**KEYOBJ**

---

---

By: Greg Nuyens

Supported by Jan Pedersen (Pedersen.pa@Xerox.com)

KEYOBJ provides a LISP imageobject which mimics a key. The default image looks like this:



These keys are pressed by clicking the mouse inside the key's image. The result of pressing a key is determined (just like the physical key) by the Interlisp-D system function KEYACTION. To enter a KEYOBJ into TEdit type ↑ o. Inside the window that pops up, call the following function:

(KEYOBJ.CREATE KeyName KeyLabel Abortable) [Function]

KeyName is the key that you want the object to behave like. (CENTER in the example above). KeyLabel is an optional label other than the key whose action it mimics. If KeyLabel is a list of two elements, the first is displayed above the second. Abortable is a flag which indicates that no transitions should be generated if the mouse button is released outside the key image.

KEYOBJ.FONT [Variable]

Determines the font in which the label is created inside the keyobj. Default is Helvetica 10.

---

---

**KINETIC**

---

---

By: Anon.

Recompiled for Medley by Larry Masinter (Masinter.PA@Xerox.COM)

**INTRODUCTION**

An ancient graphics hack, converted to work with idle.

**OPERATION**

(KINETIC WINDOW)

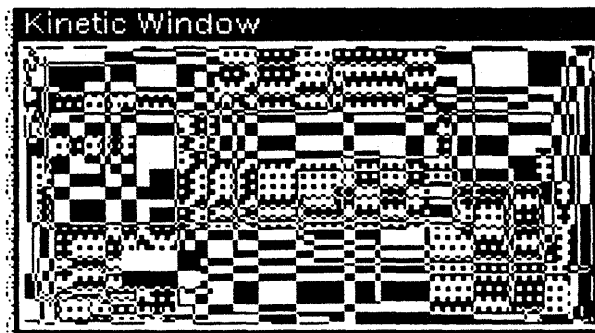
[Function]

to randomly invert rectangles on *WINDOW*, or on KINETICWINDOW (set up first time). Choosing the Kinetic on the Idle Choose Display menu will select the KINETIC function as the Idle display.

CHECKSHADE

[Variable]

If non-nil, CHECKSHADE is a texture which is used for some of the rectangles sometimes. Defaults to 63903.



---

---

**KOTOLOGO**

---

---

By: Masinter (Masinter.PA@Xerox.COM)

Uses: none

This document last edited on August 17, 1988.

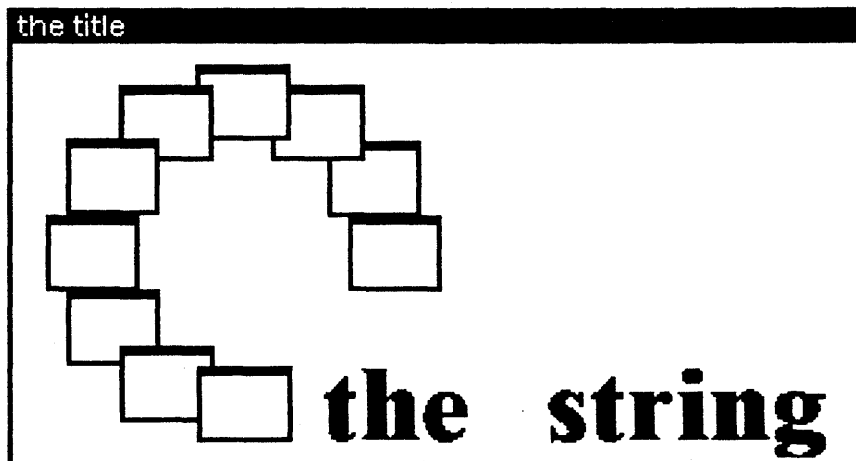
**INTRODUCTION**

Makes a Koto-style logo window.

(KOTOLOGOW *string where title angledelta*)

[Function]

Works like LOGOW did in Koto. Put *string* as the main logo name, with *title* in the window title. *angledelta* is the angle at which the little windows go. *where* is either a position or an old window. For example (KOTOLOGOW "the string" NIL "the title" 30) produces:



---

---

**LIFE**

---

---

By various folks, including help from Mike Dixon (MikeDixon.PA@Xerox.COM) and Larry Masinter (Masinter.pa@Xerox.com)

This Life program is a translation of the Smalltalk-80 version in the book **Smalltalk-80: The Language and its Implementation**, by Goldberg and Robson.

Input is a window where the "on" pixels are interpreted as living cells. The window is continually updated as life goes on.

Now an "idle" hack: LIFEDEMO as a display function plays life with the bits of the screen (in a copy of them in a window, e.g., it doesn't smash your screen.)

(Lifeldle W N)

[Function]

Run Life in window W, using the bits behind W as a starting point. N is optional, and can either be 1, 2, 4 or 8. Its the magnification of the life window.

(Life W N)

[Function]

Like Lifeldle but uses the current contents of the window.

---



---

## LOADMENUITEMS

---



---

By: sML (Lanning.pa@Xerox.com)

### INTRODUCTION

Some utility files are so useful that users will always want them in their system: these files are typically loaded from the users INIT file. A (rather large) number of other utilities are only sometimes useful. Users are faced with the choice of either loading these files from their INIT files (slowing down the initialization process and consuming space, whether the utility is needed or not) or having to remember how to load and initialize these files.

LOADMENUITEMS addresses this problem: it defines a new filepackage command that can be used to add entries onto the background menu for easy loading of utility files.

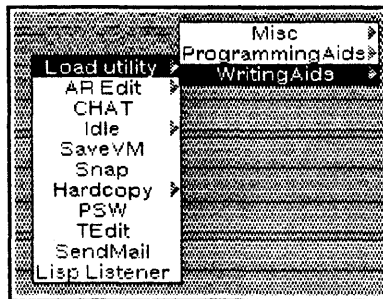
[NOTE: All (advertised) symbols in this utility are in the INTERLISP package.]

### EXAMPLE

The filepackage command

```
(COMS
  ;; Make it easy to load some oft-used utilities
  (FILES LoadMenuItems)
  (LOADMENUITEMS WritingAids Sketch VirtualKeyboards ProofReader)
  (LOADMENUITEMS ProgrammingAids (Spy (SPY.BUTTON)))
  (LOADMENUITEMS NIL VStats Calendar))
```

will add an entry "Load utility" to the background menu. "Load utility" will have three subitems: Misc, ProgrammingAids, and WritingAids:



WritingAids will in turn have three subitems: ProofReader, Sketch, and VirtualKeyboards; Misc will have the subitems Calendar and VStats; ProgrammingAids will have the single subitem Spy.



Selecting any of these final menu items will load the corresponding file. In addition, the Spy menu item will evaluate the form (SPY.BUTTON) after loading the file Spy.

## INTERFACE

(LOADMENUITEMS *group utilDescr1 utilDescr2 ...*) [FilePackageCommand]

Dumps out to the file a form that will add items to the background menu for loading *utilDescr1*, *utilDescr2*, ... Each item will be added to the *group* subitem of the "Load utility" item on the background menu; if *group* is NIL it defaults to "Misc".

In the simplest case, *utilDescr* is a LITATOM. This is used when you want to load a file without any extra initialization, and the file is on one of the directories in DIRECTORIES. Selecting the resulting item will evaluate (DOFILESLOAD '*utilDescr*') and print an informative message in the prompt window when the DOFILESLOAD is finished. The added item will have the label *utilDescr*.

In the general case, *utilDescr* is a list. This is used when you want to specify an initialization form to be evaluated when the utility is loaded, or when the file description is not a LITATOM. In this case, selecting the menu item will evaluate (DOFILESLOAD (CAR '*utilDescr*')). If *utilDescr* is a list of two elements, the CADR of *utilDescr* will be evaluated after the utility is loaded; otherwise an informative message will be printed in the prompt window. The added item will have as a label the first LITATOM in the CAR of *utilDescr*; this is the first file that will be loaded when the item is selected.

In each of the above cases, the item is removed from the background menu after the utility is loaded and initialized.

When a utility is loaded from the "Load utility" menu, the event is added to the history list. This way you can UNDO loading a utility.

Some illustrative examples:

```

;; This adds the item "VStats" to the "Misc" subitem
(LOADMENUITEMS NIL VStats)

;; Selecting the "Spy" item will load SPY and call
;; SPY.BUTTON to bring up the spy button icon
(LOADMENUITEMS ProgrammingAids (Spy (SPY.BUTTON)))

;; This will add the item "GO" to the "Games" group
(LOADMENUITEMS Games (((SYSLOAD FROM {PHYLUM}<Foster>Lisp) GO)))

;; These items are useful for Lafite users, but aren't always
;; needed
(LOADMENUITEMS MailTools LafiteFind Undigestify MailScavenge)

```

## FUNCTIONS

(AddLoadMenuItem *group fileDescr startUpForm*)

[Function]

Add a menu item to the background menu that will load the files. The item will be added under the top level item "Load utility". *group* is the submenu name for this file; the default is `Misc`. *fileDescr* is a list that can be passed to `DOFILESLOAD` to load the files. *startUpForm* is an optional form that will be evaluated after the `DOFILESLOAD`; the default will print a nice message in the prompt window. The `LOADMENUITEMS` filepackage command described above expands to calls to `AddLoadMenuItem`.

`AddLoadMenuItem` is UNDOable.

(PickLoadUtilityItem *utility-name* &OPTIONAL *group-name no-errors-p*)

[Function]

This is the programatic equivalent of selecting the item named *utility-name* from the "Load utility" item on the background menu. If *group-name* is given, only that group under the "Load utility" item is searched for the utility; otherwise the entire menu item is searched. If multiple matching items are found, a continuable error is signaled. Proceeding from this error will let you pick one of the items to execute. If no matching items are found, a continuable error is signaled. The *no-errors-p* flag controls whether or not these errors are actually signaled: if *no-errors-p* is true, `PickLoadUtilityItem` ignores the errors. `PickLoadUtilityItem` return T if the utility was loaded, NIL otherwise.



---



---

**LOGIC**


---



---

By: Roberto Ghislanzoni ("Roberto\_Ghislanzoni".MKTRXI@Xerox.com)

Uses: TABLEBROWSER

This document last edited on 19-Sep-88 14:03:58.

## INTRODUCTION

This package is devoted to people who want to use a logic paradigm in their programming environment. LOGIC was initially developed in Franz Lisp at the Computer Science Department of the University of Milan: now a modified part of its kernel is running in Common Lisp, so it is possible to use it under every machine running CL: within the Lisp environment, some features are available in order to ease the construction of the programs.

All of the source codes are available: sorry if they are awful! But it's better to have efficiency than syntactic sugar ...

## LOGIC MANUAL

LOGIC is essentially a theorem prover based on Horn clauses: the user is allowed to create many theories and within these theories to specify some predicates (clauses); as FOL does, it is also possible to specify some *semantic attachments* (SA), in order to use all the capabilities of the environment: in our implementation, these SAs are expressed in Lisp. A goal proof is performed within specified theory(es); the user is allowed to dynamically change the theories involved.

These are the elements of the language:

- a *variable* is an atom beginning with '?'
- an *atomic formula* is a list beginning with the name of the predicate and followed by the terms: (on table book)
  - (mother ?x ?y)
- a *clause* is a list beginning with the consequent, followed by the special symbol ':-', and by the sequence of the antecedents:
  - ((grandfather ?x ?y) :- (father ?x ?z) (father ?z ?y))
- a *set of clauses* is the definition of a predicate
  - ((append () ?a ?a) ((append (?a . ?b) ?c (?a . ?d)) :- (append ?b ?c ?d)))
- a *theory* is a set of the definitions of some predicates

**HOW TO LOAD AND INIT LOGIC**

In order to use LOGIC, load the files LOGIC and UNIFIER. From within the Envos Lisp Environment, you can also load the development environment LOGIC-DEVEL.DFASL. After loading the files, call the functions:

- (CREATE-BACKGROUND-THEORY): this function creates the main theory reading the data it needs from the file MAIN.LGC.
- (CREATE-VARIABLES): this functions creates and initializes the variables used by the unificator; it takes a few time to perform its job.

These are the functions available from the top-level executive of Lisp:

(ALL VARS CONJ THS ) [Function]

Returns the *vars* that satisfies the goal (*conj*) in the list of theories (*ths*): the background theory is always used. For example you can ask the system to prove:

```
(ALL '(?a ?b) '((append ?a ?b (1 2 3))) '(append-theory))
--> ((NIL (1 2 3)) ((1) (2 3)) ((1 2) (3)) ((1 2 3) NIL))
```

(ANY HOW-MANY VARS CONJ THS ) [Function]

Returns *how-many vars* that satisfies the goal:

```
(any 2 ?a '((append ?a ?b (1 2 3))) '(append-theory))
--> (NIL (1))
```

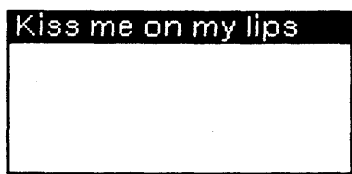
(ATTACH SA-NAME DEFINITION THEORY-NAME) [Function]

Allows to create semantic attachments:

```
(ATTACH 'createw '(lambda (name) (IL:CREATEW () name)) 'my-theory)
```

and now:

```
(ANY 1 () '((createw "Kiss me on my lips"))) '(my-theory)
-->
```



(CREATE-THEORY THEORY-NAME) [Function]

Creates a brand-new theory with that name: return the name of the theory created, not the theory itself.

(LIST-ALL-THEORIES) [Function]

Return a list of the defined theories, currently available.

(LOAD-THEORY THEORY-NAME) [Function]

Loads from the current directory the specified *theory-name*; the name of the theory file has the extension .LGC, and it must be previously created by the corresponding function SAVE-THEORY

(LOGIC-ADDA *PRED CLAUSES THEORY-NAME*) [Function]

Adds to the definitions of the predicate *pred* the specified *clauses*, that holds in the theory *theory-name*: the clauses are put in front of the already existing clauses:

(LOGIC-ADDA 'C'(((C 1)) ((C ?x) :- (A ?x))) 'my-theory)

(LOGIC-ADDZ *PRED CLAUSES THEORY-NAME*) [Function]

Adds to the definitions of the predicate *pred* the specified *clauses*, that holds in the theory *theory-name*: the clauses are put at the end of the already existing clauses:

(LOGIC-ADDZ 'C'(((C 2)) ((C ?x) :- (A ?x) (B :y))) 'my-theory)

(LOGIC-ASSERT *PRED CLAUSES THEORY-NAME*) [Function]

Replaces all previous definitions of the predicate *pred* with *clauses*.

(LOGIC-DELETE *PRED-OR-SA THEORY-NAME*) [Function]

Erases from the theory *theory-name* the definition of *pred-or-sa*, that may be either a predicate or a semantic attachment

(LOGIC-DELETE-FACT *FACT-NAME FACT-CLAUSE THEORY-NAME*) [Function]

Erases from the definition of the clauses on the predicate *FACT-NAME* the specified clause *FACT-CLAUSE*, within the theory *THEORY-NAME*.

(MERGE-THEORIES *NEW-THEORY-NAME &REST LIST-OF-THEORIES*) [Function]

Creates the new theory *NEW-THEORY-NAME* made up by all the predicates and sas that hold in all the theories *LIST-OF-THEORIES*: now no control is performed on the consistency in the merging of the theories

(PROVE *CONJ THS*) [Function]

Calls the prover on the specified goals *conj*. *THS* is a list of the theory(es) used. PROVE returns only T or NIL

(SAVE-THEORY *THEORY-NAME*) [Function]

Writes on the local directory the contents of the theory *theory-name*. You will find later a file whose name is composed by the theory name and the extension LGC.

The format of the contents of the file is the following:

*theory-name*

number of semantic attachments

<sa name1> <sa definition>

..

<sa nameN> <sa definition>

number of predicates

<predicate name 1> <clauses for predicate 1>

..

<predicate name N> <clauses for predicate N>

A theory file (with .LGC extension) may be created by the user employing a text editor like Emacs or VI (on Symbolics, SUNs etc.), avoiding the saving of the theory every change he performs.

(SHOW-DEFINITION *ELEMENT THEORY-NAME*) [Function]

Shows the definition of *element*, either a predicate or a semantic attachment.

(SHOW-THEORY *THEORY-NAME* &OPTIONAL VERBOSE) [Function]

Shows the contents (name of predicates and sas) of the theory *theory-name*; if *verbose* is T, all the definitions are shown.

### THE BACKGROUND THEORY

In the background theory, many interesting primitive predicates are available:

**!** [Predicate]

The cut predicate, well-known to the PROLOG programmers: a typical example of its use can be the definition of the predicate NOT:

```
((not ?formula) :- (wff ?formula) ! (fail))
((not ?formula)))
```

(TRUE) [Predicate]

This predicate always succeeds

(FAIL) [Predicate]

The predicate that never succeeds

(PRINT ?arg) [Predicate]

Prints the argument ?arg passed by

(EVAL&PRINT ?arg) [Predicate]

This predicate evaluates and print the result of evaluation of the form ?arg:

```
(prove '((eval&print (+ 3 4))) '(my-theory))
```

7

T

(LOGIC-ADDA ?PREDICATE-NAME ?CLAUSES ?THEORY-NAME) [Predicate]

Adds in front of the clauses that define the predicate ?PREDICATE-NAME in the theory ?theory-name the other clauses ?CLAUSES

(LOGIC-ADDZ ?PREDICATE-NAME ?CLAUSES ?THEORY-NAME) [Predicate]

Adds to the end of the clauses that define the predicate ?PREDICATE-NAME in the theory ?theory-name the other clauses ?CLAUSES

(LOGIC-ASSERT ?PREDICATE-NAME ?CLAUSES ?THEORY-NAME) [Predicate]

Replaces all definition for the predicate ?PREDICATE-NAME in the theory ?THEORY-NAME with the new clauses ?CLAUSES

(LOGIC-DELETE ?PREDICATE-OR-SA--NAME ?THEORY-NAME) [Predicate]

Deletes all definition for predicate (or sa) ?PREDICATE-OR-SA-NAME in the theory ?THEORY-NAME

(LOGIC-DELETE-FACT ?FACT-NAME ?FACT-CLAUSE ?THEORY-NAME) [Function]

Erases from the definition of the clauses on the predicate *FACT-NAME* the specified clause *FACT-CLAUSE*, within the theory *THEORY-NAME*.

**(SET ?var value)** [Predicate]

With this predicate it is possible to set a variable within the demonstration (remind that a variable always starts with a '?'):

```
(prove '((set ?x (list 'a 'b 3))(print ?x)) '(my-theory))
```

```
--> (a b 3)
```

```
T
```

**(RETRACT ?theory-name)** [Predicate]

Tells the interpreter that it must use no more the theory *?theory-name* during the ongoing demonstration; this elision is made only on the current active node of the demonstration tree

**(USE-THEORY ?theory-name)** [Predicate]

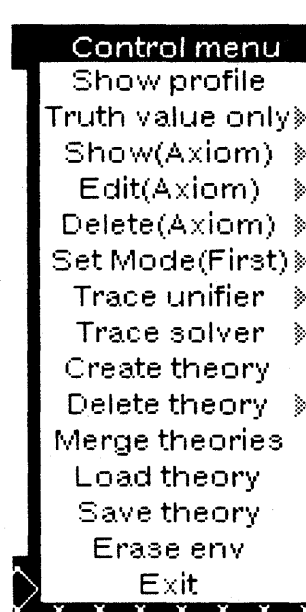
Tells to the interpreter that it must use the theory *?theory-name* for the ongoing demonstration.

**(WFF ?formula)** [Predicate]

This is a second order predicate that allows you to prove the truth value of the well formed formula *?formula*

If you load only the LOGIC files, this is the environment you have. On Xerox machines, you can also load the file LOGIC-DEVEL, that allows you to have the development environment: a new entry in your background menu is created, and so you are able to open a logic demonstration window.

This is the control menu:



**SHOW-PROFILE:** shows the current profile: the **MODE** of demonstration (**FIRST**, **ALL**, **INTERACTIVE**), and the tracing flags on unifier and solver

**TRUTH VALUE ONLY:** this flag controls if the prover returns all the goals with the variables instantiated or only the values **T** or **NIL**

SHOW AXIOM: shows the definition of an axiom or of a semantic attachment

EDIT AXIOM: edits the definition of an axiom or of a semantic attachment

DELETE AXIOM: deletes the definition of an axiom or of a semantic attachment

SET MODE: sets the mode of the demonstration: this may be ALL, FIRST or INTERACTIVE

TRACE SOLVER: the solver is the procedure of the interpreter that takes as arguments a tree, a formula and the clauses for that formula, and gives back the new tree obtained by the resolution operation; its behaviour is traced on a debugging window which has the middle menu capability of dribbling; the output file has the extension TRC.

TRACE UNIFIER: traces the going on of the unifier on a debugger window; this window too has the middle menu capability of dribbling its output. The pattern, the datum and the unification environment will be shown to the user.

CREATE THEORY: creates a new theory

DELETE THEORY: all the theories loaded are showed in a tablebrowser at the left of the main window; when you select one or more theories, this means that you want to use them for your demonstration; this command deletes the selected theories; you can however undelete or expunge them with the subitems of this entry. Remember that, for undeleting the selected theories with the tablebrowser mark(▶), you must click middle button on it and press CTRL (PROP) key

MERGE THEORIES: merges the selected theories in a new theory; the user is prompted for the name of the new theory

LOAD THEORY: loads a theory from a file in the current directory

SAVE THEORY: save the selected theory(ies) on the corresponding files

ERASE ENV: erases all the environment of the window

EXIT: exits from the environment

Remember that, for every demonstration requested, there must be at least one theory selected in the tablebrowser at the left of the main window

I hope these notes help you to use LOGIC.

You can find some examples in the theory file EXAMPLES.LGC.

Any suggestion is welcome: since it is not fully tested, please notify every kind of error or bug you will find.

### EXAMPLES

Choose LOGIC from the background menu: a new window will appear: choose LOAD THEORY from the control menu and type in EXAMPLES at the request in the prompt window: mark the theory loaded in the theories window and try:

```

NIL
12 Logic>((APPEND ?A ?B (1 2 3)))

```

the system will respond you

```
12 Logic>((APPEND ?A ?B (1 2 3)))
((APPEND NIL (1 2 3)
  (1 2 3)))
NIL
13 Logic>
```

Click now on SHOW PROFILE: you will see

```
Mode: FIRST /Unifier: NOTRACE /Solver: NOTRACE /Values: NIL
```

Choose SET MODE ALL (submenu) and retry the same goal as before: you get the answer:

```
13 Logic>((APPEND ?A ?B (1 2 3)))
((APPEND NIL (1 2 3)
  (1 2 3)))
((APPEND (1)
  (2 3)
  (1 2 3)))
((APPEND (1 2)
  (3)
  (1 2 3)))
((APPEND (1 2 3)
  NIL
  (1 2 3)))
NIL
```

In the theory EXAMPLES a simple little maze is described: type in the goal:  
 ((search a g))  
 that will find a path from the room 'a' to the room 'g'.

M	I	E	F
B	C	D	H
A	N	G	L

Here are other examples of goals you can try:

```
((sa-member 3 (1 2 3 4 5)))
```

```
((logic-member 3 (1 2 3 4 5)))
```

The first one is a SA, the latter is a predicate.

```
((NOT (A 1))) --> T
((NOTMEMBER 2 (1 3 4))) --> T
and so on.
```

Try now all the other features of the language.

You can ask the system for the same goals from the lisp listener:

```
(load-theory 'examples)
(prove '((APPEND ?X ?Y (1 2 3 4)) '(examples)) --> T
(all '(?X ?Y) '((APPEND ?X ?Y (1 2 3 4)) '(examples))
--> ((NIL (1 2 3 4)) ((1) (2 3 4)) ((1 2) (3 4)) ((1 2 3) (4)) ((1 2 3 4) NIL))
```

Have fun!



---



---

## LOOKUPINFILES

---



---

By: `dgb (Bobrow.pa@Xerox.com)`

### INTRODUCTION

The LOOKUPINFILES module is a facility for building quick and easy access to on-line files. It allows search for a target string through all files in a specified list. It finds the target, and brings up the file in a window, with the target selected in inverse video. The file can then be used as the source for text for other documents. It is the basis for the user facilities of ADDRESSBOOK and FIND-CITATION. Its interface is defined by the function:

`(MakeLookupWindow fileList processName mainWindowRegion iconBM iconMask iconPosition)`

These arguments are used as follows:

<code>fileList</code>	List of file names. Search goes through these files in order
<code>processName</code>	Name appearing in PSW for this lookup process
<code>mainWindowRegion</code>	Region for window showing text found
<code>iconBM</code>	Bit map for icon when mainWindow is shrunk
<code>iconMask</code>	Mask for icon
<code>iconPosition</code>	Position for icon

Arguments other than `fileList` are optional. Calling `MakeLookupWindow` will construct a Lookup window, and shrinks it to the icon provided. Opening this icon shows the window interface to the search process. To find any string in one of the files, type the string followed by a return. The program will quickly search through the files and show you an occurrence of the string typed. The located string is shown in inverse video. The title of the window will contain the name of the file in which the entry was found. The search ignores case; e.g. "bobrow" matches "Bobrow". The text of the document is scrollable, and any portion can be shift selected into another document.

Type carriage return, or click on **Next Occurrence** to search further in the files for the same string. If no (further) occurrences are found, the text window will display a message indicating the failure. Searching again after failure will start the search from the beginning of all the files, using the same lookup string. Typing a new string can be repeated as many times as you like. When you are done, just SHRINK the window back to its icon.

The window below is taken from the use of this package as an online address book.

---

<b>Lookup String: stefik</b> <b>Lookup String:</b>
<b>Next Occurrence</b>
<b>Looking in: {phylum}&lt;bobrow&gt;lisp&gt;addresses.ted</b>
Dr. Mark J. <b>Stefik</b> Xerox Palo Alto Research Ctr. Knowledge Systems Area 3333 Coyote Hill Road Palo Alto, CA 94304 Residence:

---

Example ADDRESSBOOK window

---

## NOTES

### Caching Files

When you first open the window, the program will copy the files to {CORE}, significantly speeding up queries. Bugging in the title of the main window with the left or middle mouse button will produce a menu with an option to recache all these files.

### Editing Your Files

To edit the file in which a string is found, bug in the title of the main window, and select the option "Edit File". You will be requested to confirm that you want to edit the file. If you confirm, a TEDIT process editing the file will be set up. This process is independent of the lookup process. To make editing changes visible to the lookup process, PUT the file in TEDIT; when it is done, recache the file in core. To recache just the file edited, (the one specified in the title bar of the window), select the option "Recache just this file" in the title bar menu. You can recache all files by selecting that option in the title menu.

### Adding to the List of Files

To add to the list of files being used for lookup, select the option "Add new file" in the title bar menu. This file will be added to the list of files to be searched, and cached in core.

---

---

**MAGNIFIER WINDOW**

---

---

By: Richard R. Burton (Burton.PA @ Xerox.Com)

This document last edited on March 17, 1986.

**INTRODUCTION**

File: MAGNIFIER.LCOM

Tired of giving demos in which only the two people sitting next to you can see the screen? This small package implements magnifying windows, windows that show an enlarged copy of that portion of the screen that is around the cursor. A magnifying window can be created either by calling the function MAGNIFYW or by selecting the item "Magnifier" from the background menu. A magnifying window can be made to any size and is distinguished by its large border. Once a magnifying window has been created, it can be activated by clicking the left button in it. While activated, the cursor will be replaced by a black rectangle and the contents of the rectangle will be displayed in the magnifying window enlarged by a factor of 4. The contents will continue to track the location of the cursor until the left button is clicked a second time. The magnifier can be reshaped.

**Suggested use:** When six people drop into your office unannounced for a demo, create a magnifying window across the top or bottom of your screen (so the people in the back can see it easily). When it is important for people to read what you are talking about, move the cursor into the magnifier, click the left button, move the cursor over the area of interest and, when the image in the magnifier has what you want, click the left button again. This will leave an enlarged part of the screen in the magnifier and free the mouse of other things. You can leave magnifier active but it will not block (so no other processes get to run) and if you move the cursor, the image in the magnifier will move too.

---



---

## MakeGraph

---



---

By: D. Austin Henderson, Jr.

Supported by: Nick Briggs (Briggs.pa@Xerox.com)

### INTRODUCTION

MakeGraph is a module which sits on top of Grapher and helps one create graphs depicting a data structure by walking through it. The central idea is that each point in the walk (and node in the graph) is characterized by a datum/state pair and motion is defined by a graph specification in the form of state transition function. This function is specified by a collection of state specifications, each of which indicates how to display (label and font) the datum when one is in that state and how to find the datum/state pairs which are the sons of that node. Also the state specification may specify additional roots for the walk. The generation of a branch of the graph ceases when either there are no sons of a node, or an already encountered node is revisited (identical datum and identical state). The module contains a function for creating such graphs and an example of its use: a function which graphs the graph specifications themselves. Comments are welcomed

### FUNCTIONS

The main functions are:

**(MAKE.GRAPH *WINDOW TITLE GRAPH.SPECIFICATION ROOTS CONTEXT*  
*LEFTBUTTONFN MIDDLEBUTTONFN TOPJUSTIFYFLG DEPTH*)** [Function]

Creates a MAKEGRAPH window. If *WINDOW* is NIL, then a new one will be created and the user will be prompted to position it. Otherwise, the graph will be shown in *WINDOW*. The window will be titled with *TITLE*, will call *LEFTBUTTONFN* and *MIDDLEBUTTONFN* on nodes selected (or NIL if selection is made where no node is positioned), and will be justified as indicated by *TOPJUSTIFYFLG* (a la Grapher). The button functions are defaulted to MAKE.GRAPH.LEFTBUTTONFN (which scrolls the window so that the selected node is in the middle of the window, or if the left shift key is depressed, prints out information about it) and MAKE.GRAPH.MIDDLEBUTTONFN (which provides a menu of two choices: INSPECT - inspects the datum of the node selected, or if the left shift key is depressed, inspects the node itself; and SUB.GRAPH - which opens another MAKEGRAPH window with the same parameters as this one, but with graph starting at the selected node). The arguments to MAKE.GRAPH are added as properties to the window under their argument names. Selecting in the title invokes the functions which are the values of the window properties TITLE.LEFTBUTTONFN and TITLE.MIDDLEBUTTONFN (not in the calling sequence; set by the user if desired; called with a single argument - *WINDOW*; defaulted to a function which provides a menu of UPDATE and SHOW.GRAPH.SPEC (see functionality below)). The graph is created according to the *GRAPH.SPECIFICATION* (see below) to depth *DEPTH*, starting from *ROOTS* which are (DATUM . STATE) pairs. *CONTEXT* is an extra argument which is a passed along to all accessing expressions.

**GRAPH.SPECIFICATION** [Parameter]

A GRAPH.SPECIFICATION is a property list of STATE.SPECIFICATIONs where the properties are the state names.

## STATE.SPECIFICATION

[Property list]

A STATE.SPECIFICATION is a property list whose properties and values are as follows (in this, EXPR means a LISP form which will be evaluated in an environment in which DATUM is bound to the node's datum, STATE to the node's state, and CONTEXT to context):

## LABEL

[Property]

an expression returning something which will be printed as the label of the node; if no LABEL property is provided, the string "???" will be used.

## FONT

[Property]

an expression returning the font to be used for this node; if no FONT property is provided, the default font for the grapher will be used.

## SONS

[Property]

a form indicating a list of (DATUM . STATE) pairs to be used in generating the sons of this node; the acceptable forms are any of the following:

(data-expression state-expression)

[Property value]

where data-expression returns a list of datum's for the son nodes, and state-expression is evaluated in the context of each of these in turn to produce the corresponding state of each.

(LIST (data-expression state-expression) ...)

[Property value]

a template of expressions which are evaluated individually to produce a list of sons of the same form, viz. (DATUM . STATE) pairs.

(EVAL expression)

[Property value]

the expression returns a list of (DATUM . STATE) pairs of the sons.

(UNION sons-spec ...)

[Property value]

where each sons-spec is any of these forms (recursively).

(TRACE sons-spec)

[Property value]

a device for helping debug graph specifications; the value is the value of sons-specs; the user is given the chance to INSPECT them after they have been generated.

## ROOTS

[Property]

like SONS, except the resulting (DATUM . STATE) pairs are used as possibly additional roots of the graph.

(MAKE.GRAPH.CONSTRUCT *GRAPH.SPECIFICATION INITIAL.ROOTS CONTEXT DEPTH*) [Function]

This is the functional heart of MAKE.GRAPH broken out for those who wish to handle their own interactions with grapher and the window package. It produces a list of graphnodes with labels and sons as specified by *GRAPH.SPECIFICATION* (see MAKEGRAPH), starting from *INITIAL.ROOTS* which are (DATUM . STATE) pairs. *CONTEXT* is an extra argument which is a passed along to all accessing expressions. Returns the list of graphnodes.

(MAKE.GRAPH.FIND.ROOTS *GRAPH.SPECIFICATION INITIAL.ROOTS CONTEXT DEPTH*) [Function]

Finds the real roots from a set of initial roots, using the same processing as MAKEGRAPH uses. This is helpful when you want to hand a "correct" set of roots of a structure to MAKEGRAPH without having to explore the dependencies within that structure. As with MAKEGRAPH, the data structure is processed according to the *GRAPH.SPECIFICATION* (see MAKEGRAPH), starting from *INITIAL.ROOTS* which are (DATUM . STATE) pairs. *CONTEXT* is an extra argument which is passed along to all accessing expressions. Returns the real roots as a list of (DATUM . STATE) pairs.

### Supporting Functions

(MAKE.GRAPH.UPDATE.WINDOW *WINDOW*) [Function]

Uses the window properties (which may have been changed) to reinvoke MAKE.GRAPH on the window.

(MAKE.GRAPH.SHOW.SPEC *GRAPH.SPECIFICATION*) [Function]

Uses MAKE.GRAPH to produce a graph of a *GRAPH.SPECIFICATION*. It uses as the graph.specification for this layout the value of the variable MAKE.GRAPH.SPEC.SPEC which presents *GRAPH.SPECIFICATION* (reflectively) as a graph.specification. (MAKE.GRAPH.SPEC.SPEC can serve as a template for other graph.specifications. It is a fairly complex 9-state specification. For a simpler example see below under MAKE.GRAPH.SHOW.LIST.)

(MAKE.GRAPH.EXAMPLE.1) [Function]

Calls MAKE.GRAPH.SHOW.SPEC on MAKE.GRAPH.SPEC.SPEC.

(MAKE.GRAPH.SHOW.LIST *OBJECT*) [Function]

Uses MAKE.GRAPH to produce a graph of an arbitrary Lisp object. It uses as the graph.specification for this layout the value of the variable MAKE.GRAPH.LIST.SPEC which presents *OBJECT* as a tree whose nodes are LISTPs and whose leaves are non-LISTPs.

MAKE.GRAPH.LIST.SPEC [Variable]

MAKE.GRAPH.LIST.SPEC is the simple 1-state specification below, included here as an example of a graph.specification

```
(OBJECT ( DOC (ANY LISP OBJECT)           - some documentation
          LABEL (COND ((LISTP DATUM) "( ")
                    (T DATUM))
          SONS ((COND ((LISTP DATUM) DATUM)
                 (T NIL))
              (QUOTE OBJECT))
          )
)
```

For a more complex example see above under MAKE.GRAPH.SHOW.SPEC.

(MAKE.GRAPH.EXAMPLE.2) [Function]

Calls MAKE.GRAPH.SHOW.LIST on MAKE.GRAPH.LIST.SPEC; that is, produces a graph of this simple graph.specification as a list. Notice that selecting the title command UPDATE in this window will yield a different graph of the same structure, viz. as a GRAPH.SPECIFICATION.

Other useful functions

**(MAKE.GRAPH.DATUM *NODE*)**

**[Function]**

Returns the **DATUM** associated with the graph node *NODE*.

**(MAKE.GRAPH.STATE *NODE*)**

**[Function]**

Returns the **STATE** associated with the graph node *NODE*.

**(MAKE.GRAPH.FATHER *NODE*)**

**[Function]**

Returns the graph node which is the father of the graph node *NODE*.

---

---

**MANAGER**

---

---

By: Jay Ferguson, Larry Masinter and Andrew Cameron III

Maintained by Ron Fischer (Fischer.pa@Xerox.com)

Uses: MASTERSCOPE

**INTRODUCTION**

In its latest incarnation Manager supports MasterScope and improves its performance.

**USING MANAGER:**

Manager provides a way to perform most common File Manager operations onscreen using menus, both pop-up and permanent. Activity centers around the filelst, or main, menu, and menus of items of a type in the file (like all FNS, or all VARS).

Printing and interaction is done through the Manager Command Activity Window. The first time it is needed you'll be prompted to size it onto the display. Thereafter, it will be used as needed. If shrunken before use it will wait 10 seconds after an operation and then shrink down again.

**The FILELST menu:**

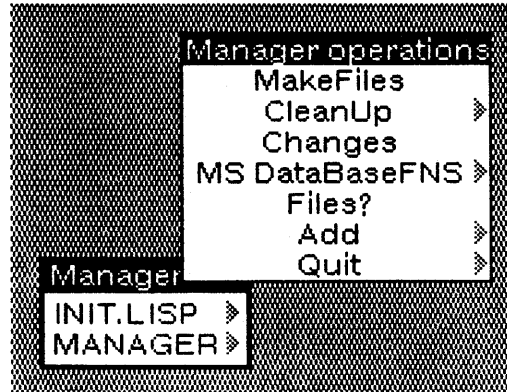
The manager provides a menu of the FILELST:





The names in the FILELST menu can be copy selected.

Middle buttoning on the title bar of the FILELST menu pops up a menu of operations which are applied to all loaded files:



These operations are the same as the similarly named functions in the File Manager interface, except for the following slide off options:

CleanUp:

Set default: TCOMPL, the default compiler will be TCOMPL.

Set default: CL:COMPILE-FILE, the default compiler will be CL:COMPILE-FILE.

MS DataBase FNS:

*various MasterScope database flags can be set*

Add, notice a file via:

LOADFNS

LOADFROM

LOAD

ADDFILE\*

Edit FILELST, edit the FILELST directly in a lisp editor window.

Quit:

Quit\*, shut down the Manager, all menu caches cleared, windows closed.

Reset, shut down and turn on the Manager again.

Left buttoning on a file in the FILELST menu (without sliding off) pops up a menu of operations on that file:



**See:**

**fast\***, prints the source of the file.  
**scrollable**, displayed in a scrollable TEdit window.

**(Re)Load:**

**Load\***, use current DFNFLG settings.  
**Sysload**, load with File Manager turned off.

**MakeFile**, dump the file

**MakeFile\***, dump the source of the file by remaking it.  
**New**, dump the source without copying unchanged defs from existing file.  
**Fast**, dump source without prettyprinting (fast).  
**CommonLisp**, dump source in commonlisp format (loads common-makefile if needed).

**List**, list the source file on the default printer.

**CleanUp:**

**CleanUp\***, dump the file according to CLEANUPOPTIONS.  
 Set default compiler: TCOMPL.  
 Set default compiler: CL:COMPILE-FILE.

**MasterScope:**

**Analyze\***, analyze the fns on the file.  
**Check**, check the file for problems.  
**Show Paths**, show paths of function calls on this file.  
**DatabaseFNS**, display the database property for this file (loads databasefns if needed):  
   Set to ASK, ask about saving MS DB information.  
   Set to ON, automatically maintain MS DB information.  
   Set to OFF, do not save MS DB information.  
**Load DB**, load an existing MS DB for this file.  
**Dump DB**, dump the current MS DB for this file.

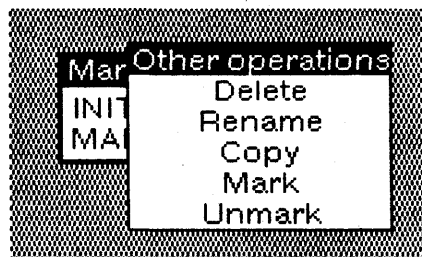
**Compile:**

**Compile\***, compile the file based on the current settings.  
 CL:COMPILE-FILE, compile the file with CL:COMPILE-FILE.

**Changes:**

**Brief\***, prints the changes that have been made to this file.  
**Everything**, prints the complete list of files changes.  
**Edit PL**, brings up a lisp editor on the file's property list.

Middle buttoning on a file in the filelst menu (without sliding off) pops up a menu of generic operations on that file:

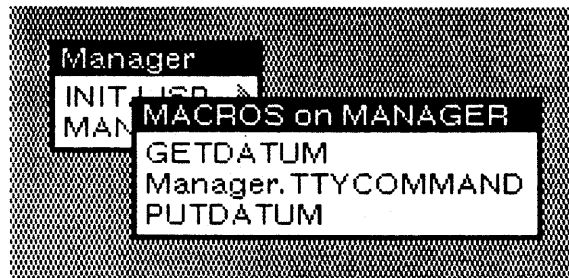


**Delete**, removes the file object from the system.  
**Rename**, prompts for a new name and renames the file.  
**Copy**, prompts for a new name and copies the file under that name.  
**Mark**, mark the contents of the file as changed.  
**Unmark**, unmark the contents of the file as changed.

Left buttoning on a file and sliding off to the right pops up a menu of types in the file:



Releasing on one of these places a menu of items of that type on the file:



This menu is not pop-up and remains on the display.

**The items of a type menu:**

These menus contain the names of all instances of a particular type on a file. Names of items in these menus can be copy selected.

Left buttoning on an item name pops up a menu of operations on that type:



Edit, brings up the source text of the item in a lisp editor.

PrettyPrint:

Show\*, prints the source text of the item quickly.

Value, prints the global value of the item's name (assumed a symbol).

Function Def, prints the global function definition of the item's name (assumed a symbol).

Property List, prints the global property list of the item's name (assumed a symbol).

Documentation:

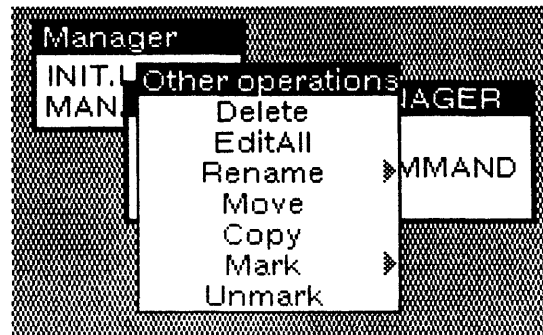
Documentation\*, prints the item's documentation string.

Describe, calls describe on the item's name (assumed a symbol).

The menu of item operations shown above is the general one. There are special menus for the following types:

FNS, FUNCTIONS, RECORDS, VARS

Middle buttoning on an item name pops up a menu of generic operations on that type:



Delete, removes this item from its file.

EditAll, edits all occurrences of this item's name in the latest source file (uses EDITCALLERS).

Rename:

Rename\*, rename this item in its file and update all uses of the name.

CopyDef, copy this item under a new name.

Rename All, rename this item in \*ALL\* loaded files.

Move, move this item into another file.

Copy, copy this item into another file.

Mark:

Changed\*, mark this item as changed by being edited.

Defined, mark this item as changed by being defined.

Deleted, mark this item as changed by being deleted.

Unmark, unmarks the source of this item as being changed (marks it "unchanged").

The file's makefile-environment has its readable argument used to bring up the lisp structure editor properly on objects in the file. When SEdit is the lisp editor, the package used depends on SEdit's "correct package" heuristic (usually that of the symbol naming what is being edited).

**Loading and controlling Manager:**

Just load the file. Manager can be started either from the background menu or by calling the FNS MANAGER (see below).

**Programmer's interface to Manager:**

(MANAGER POSITION) [FNS]

Starts up the manager. If POSITION is given, the filelst menu will be appear there.

(MANAGER.RESET RESTARTFLG) [FNS]

Shuts down the manager. If RESTARTFLG is true, manager will be immediately restarted after the shutdown.

**Manager.SORTFILELSTFLG** [INITVAR]

If true, the FILELST will be sorted, without side effecting the actual FILELST variable. If unset, defaults to T.

**Manager.MENUROWS** [INITVAR]

Maximum number of rows in a manager menu. If unset, defaults to 20.

**manager-marked-shade** [INITVAR]

The shade used to indicate that an item has been marked as changed. If unset, defaults to MENUBOLDFONT.

---

---

**MATHTONS**

---

---

By: Tad Hogg (Hogg.pa@Xerox.com)

**INTRODUCTION**

This file defines the translation array needed to convert from the Press MATH font to corresponding NS characters. This allows documents containing the MATH font to be printed on Interpress printers.

The array \MATHTONSARRAY contains the translations for most of the characters. Some may not be available on particular printers, causing them to appear as black boxes.

---

---

**MICROTEK**

---

---

By: Ron Clarke (RClarke.pa@Xerox.COM)

**INTRODUCTION**

MICROTEK is an image processing software package that enables you to operate Microtek Models-300 and 300A Intelligent Image Scanners with the Xerox 1108 and 1186 workstations. The Microtek MS-300, 300A, and MSF-300C are high-resolution optical scanners that can convert text, artwork, photographs, etc, into digital form for processing by computer. The digitized images that are output to the computer contain up to 300 black and white dots for every linear inch of the original document. Page size can be as large as 8.5 by 14 inches. Sophisticated firmware in this scanner enables the user to set the scanning area and control brightness, contrast, scaling, shading and other characteristics of the scanned images through simple commands transmitted from the 1108 or 1186. Two basic scanning modes are supported: Line Art mode for accurate capture of completely black-and-white material, and Halftone mode for faithful reproduction of material with varied shading. Mixed-mode scanning is also available.

With the MICROTEK software package you will be able to: Set the scanner to capture images of all kinds, with desired visual effects, and transmit them to the 1108/1186, save scanned images to disk, floppy or file server for later reloading to recreate images and print scanned images to a Xerox 4045 or 8044 laser printer.

**SOFTWARE REQUIRED**

MICROTEK.DFASL

MICROTEKPRINT.DFASL (if you have a Xerox 4045 or 8044 laser printer)

DLRS232C.LCOM

EDITBITMAP.LCOM

READNUMBER.LCOM

4045XLPSTREAM.DFASL (if you have a Xerox 4045 laser printer)

**FONTS USED**

MODERN 10, 12 BOLD

Other useful software for manipulating the scanned image:

Lispuser's Packages:

ACTIVEREGIONS, ACTIVEREGIONS2, AIREGIONS, FILLREGION

**HARDWARE REQUIRED**

Xerox 1108 with RS232C port (E-30 upgrade kit). It is also recommended that the 1108 have 3.5 meg of memory and a floating point processor (CPE board) to enable faster scanning and creation of bitmaps.

Xerox 1186. It is also recommended that the 1186 have 3.7 meg of memory.

Microtek MS-300, MS-300A, or MSF-300C Intelligent Image Scanner with optional serial port.

### CABLE CONFIGURATION

Note that the cable configuration is DIFFERENT for the MSF-300C scanner. Plugging a standard RS232C cable into the MSF-300C DB25 connector may result in damage to the equipment.

RS232C Port (DTE)		MICROTEK MS-300, MS-300A - DB25 Connector	
Signal	Pin	Pin	Signal
FGround	1	1	FGround
TD	2	3	RD
RD	3	2	TD
Sground	7	7	Sground

Pins 5, 6, 8 and 20 are jumpered together on the RS232C port end of the cable.

RS232C Port (DTE)		MICROTEK MSF-300C- DB25 Connector	
Signal	Pin	Pin	Signal
TD	21	3	RD
RD	9	2	TD
Ground	5,7	7	Ground

### DOCUMENTATION REQUIRED

Microtek MS-300, MS-300A, or MSF-300C Intelligent Image Scanner Operation Manual

### LOADING MICROTEK

Make sure that DIRECTORIES contains the directory where the required software is located. When the file MICROTEK.DFASL is loaded, the item "MicrotekScanner" will be added to the Background menu. If you have a Xerox 4045 or 8044 laser printer load MICROTEKPRINT.DFASL. If you have a Xerox 4045 laser printer load 4045XLPSTREAM.DFASL. Your 4045 laser printer should be connected to the TTY/DCE port.

### RUNNING MICROTEK

The process of running the Microtek scanner software consists of three phases: Scanner initialization, scanning, and creating a bitmap of the scanned image that can eventually be printed. Each of these are controlled by different menus within the Microtek Scanner Control Window.

### SCANNER INITIALIZATION

Set the Microtek scanner so that it is operating at 19200 baud by setting its internal DIP switches (See Microtek Operating Manual for details). Turn on the Microtek scanner. Select "MicrotekScanner" from the background menu and the Microtek Scanner Control window (figure 1) and Microtek Scanner Pagemap window (figure 2) will be created. (Note you may have to do a control-E and retry if cursor flashes while trying to create the control window). The scanner pagemap window is used to select the area of the image to be scanned and to select the page length. The scanner control window is used to set all other scanner parameters, start and stop scanning as well as to initiate creation and printing of scanned image bitmaps. After these windows have been created, the RS232 port will be initialized to 19200 baud and an attention



command will be sent to the scanner. If all cables are connected properly and the scanner is on, the message "MICROSCAN 300(A) V# is ready" will be displayed in the Microtek Status Window. If the cable is configured incorrectly or the scanner is not on or ready the message "Microtek Not Responding ... Check scanner and cable" will appear instead.

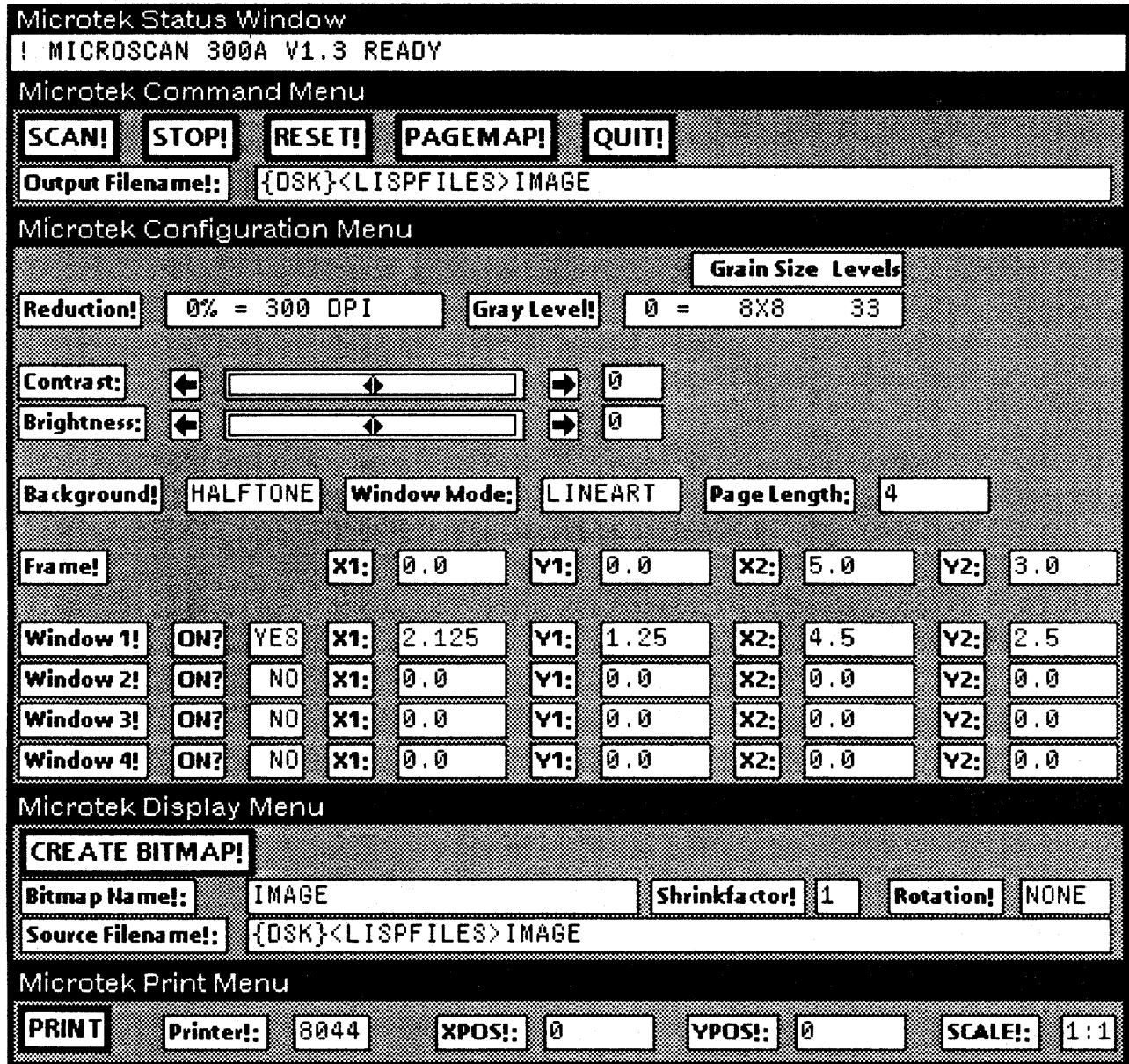


FIGURE 1 - MICROTEK SCANNER CONTROL WINDOW

Before scanning can be initiated, a number of parameters have to be set by the user via the Microtek Command Menu and Microtek Configuration Menu as follows:

**Microtek Command Menu:**

**Output FileName** Left buttoning on this item allows you enter the name of the file on disk, floppy or fileserver where the scanned data is to be saved. Be sure to type a carriage return to terminate this entry.

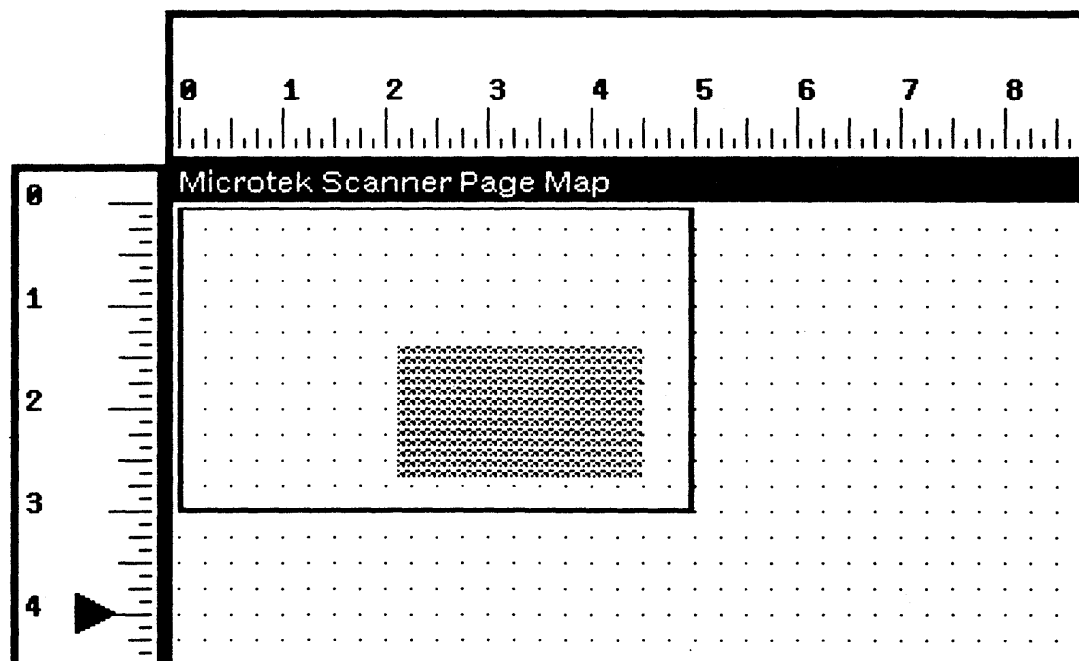
**Microtek Configuration Menu:**

**Reduction** Left button on the number next to the item Reduction and a menu will appear. Reduction can be changed from 0%, which corresponds to scanning at 300 dots per inch (DPI) to 75%, which corresponds to 75 DPI.

**GrayLevel** Left button on the number next to the item GrayLevel and a menu will appear allowing you to choose from a selection of gray levels based on grain size and number of gray levels within that grain size.

**Contrast** Left button on either the the left or right arrow to either decrease or increase the contrast setting.

**Brightness** Left button on either the the left or right arrow to either decrease or increase the brightness setting.



**FIGURE 2 - MICROTEK SCANNER PAGEMAP WINDOW**

**BackGround** Select either HALFTONE or LINEART as the primary scanning mode for the image. Line Art mode is for accurate capture of completely black-and-white material, and Halftone mode for faithful reproduction of material with varied shading.

**Pagelength** Move the cursor to the vertical ruler of the page map ( figure 2 ). The cursor will change to a right pointing triangle. Position this triangle and left-button to select the pagelength. The page length will also show up in the configuration menu. The page length should be set so

that it is longer than the actual page length of the document to be scanned. Otherwise you will get a paper jam message at the completion of scanning. The minimum page length is 3 inches and the maximum page length is 14 inches.

**Frame** The scanning frame is an area within the document that will be scanned. The maximum scanning frame is 8.5" by 14". Left button on the item Frame and you will be prompted to sweep out an area on the scanner page map to select the primary area to be scanned. The horizontal and vertical rulers and the page map grid dots can be used as a guide in determining the dimensions of the scanning frame. When the the scanning frame has been swept out, a box of the scanned area will be drawn on the page map and the actual X and Y coordinates of the top lefthand corner and lower righthand corner will appear next to corresponding items on the configuration menu (See Figure 2).

**Windows 1-4** Windows are areas within the scanning frame that are scanned in a different mode from the rest of the frame. If LINEART mode is selected as the background, all material in any windows you set will be scanned in Halftone mode, and vice versa.

The method used to set the windows is similar to that used to set the scanning frame except that you first need to specify whether the window is to be selected or not. This is done by left buttoning on the YES/NO indicator next to each window. A menu will pop-up and will allow you select "yes" or "no". After making your selection, left buttoning on the appropriate Window # will cause you to be prompted to sweep out an area within the scanning frame. Each selected window will be displayed and have a unique shade to it (See Figure 2). The only restriction is that the scanning mode must not change more than twice in one 8.5" horizontal scan line. Thus, if two windows lie across the same scan line they must extend to the edges of the page setting area. (Note that material to the left and right of the frame is scanned but not transmitted to the 1108.) You can select different windows for halftone vs lineart mode by switching between backgrounds. The item above WINDOW1 indicates which window mode is selected. An illegal window setting will result in an error message when you attempt to scan. Also note that the windows will be displayed on the scanner pagemap only if there is a "yes" next to the window number.

## SCANNING

After the Microtek scanning parameters have been initialized, the document to be scanned should be placed in the scanner top-first with the image to be scanned facing away from the user. Scanning is initiated by left-buttoning SCAN on the Microtek Command Menu. The software first creates a scratch file in {CORE} for storage of the incoming data. It then sends the scanning parameters to the scanner and if all are valid the scanning process starts as indicated by movement of the rollers. You have up to 5 minutes to insert a document before the scanner automatically stops. After scanning has been completed you will be notified in the status window that it is saving the core file to the file specified in Output Filename. It takes approxiamtely 20 minutes to scan an 8.5" x 11" document at 300 DPI.

You may stop the scanning at any time by selecting STOP. The document will be ejected and the scanner reset. You can also explicitly reset the scanner by selecting RESET. This closes the scanner scratch file if it is open, sends a reset command to the scanner and then sends the attention command. If everything is reset properly, you will get the message "MICROSCAN 300(A) V# is ready" in the status window.

## CREATING BITMAPS OF SCANNED IMAGES

The Microtek Display Menu is used to create bitmaps from a file that contains scanned data. Select SOURCE FILENAME and enter the name of the file that contains the scanned data. Select BITMAP

NAME and enter the name of a variable that you would like the bitmap bound to. **Be sure to type a carriage return to terminate the entry of each of these items.** Left button on the number next to SHRINKFACTOR and choose a factor by which you want the bitmap shrunk. The default value is 1. Left button on the item next to ROTATION and choose how you want the scanned image to be rotated. The default is "none." After these items have been set, you can then select CREATE BITMAP to start the bitmap creation process. The status window will be updated as it proceeds to create the bitmap and finally, you will be prompted to sweep out a scrollable window to display the bitmap. NOTE: Depending on the size of the bitmap, rotation may take a "very" long time and will look like your machine has frozen...be patient, it will come back. If you desire to save the bitmap(s) on a file you can do the following:

```
(SETQ filenameCOMS '((VARS bitmapname1 bitmapname2 etc))).
(MAKEFILE '{device}<directory>filename)
```

### PRINTING BITMAPS OF SCANNED IMAGES TO A XEROX LASER PRINTER

If you have the package MICOTEKPRINT loaded you will have a MicrotekPrint Menu under your display menu (See Figure 1). Select BITMAPNAME on the display menu and enter the name of the bitmap that you would like to print. To select where on the page the bitmap is printed, left button XPOS and YPOS and enter a number. For the 4045 laser printer the values of XPOS can be between 0 - 2550 and YPOS, between 0 - 3300. 1" = 300 print units o 4045. For an 8044 Interpress laser printer the values of XPOS can be between 0 - 21590 and YPOS, between 0 - 27940. 1" = 2540 Interpress units. The scale that an image is printed at is dependent upon its initial scanned reduction/DPI. You can increase or decrease the scale at which the bitmap is printed by buttoning on the number next to the item SCALE and selecting a scaling factor. On an 8044 Interpress printer a scale of 8:1 will magnify an image by 8 times on printing, 1:1 will print at true size and 1:8 reduce the image by 8 times. Values in between are also available. On a 4045 laser printer only a limited number of scale factor are availble and is dependent upon the original scan reduction as shown in the table below.

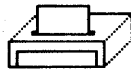
REDUCTION (%)	RESOLUTION (DPI)	SCALES ALLOWED
0	300	1:1, 2:1, 4:1
5	285	1:1, 2:1, 4:1
10	270	1:1, 2:1, 4:1
15	255	1:1, 2:1, 4:1
20	240	1:1, 2:1, 4:1
25	225	1:1, 2:1, 4:1
33	200	1:1, 2:1, 4:1
35	195	1:1, 2:1, 4:1

REDUCTION (%)	RESOLUTION (DPI)	SCALES ALLOWED
40	180	2:1, 1:1, 1:2
45	165	2:1, 1:1, 1:2
50	150	2:1, 1:1, 1:2
55	135	2:1, 1:1, 1:2
60	120	2:1, 1:1, 1:2
67	100	1:1, 1:2, 1:4
70	90	1:1, 1:2, 1:4
75	75	1:1, 1:2, 1:4

Select PRINT to initiate the printing process. NOTE: The amount of reduction that you will be able to do is dependent upon the number bits that were originally scanned in. If you make the scale too small nothing will be printed out.

#### OTHER ITEMS AND GENERAL COMMENTS

On the Microtek Command Menu, left buttoning the item PAGEMAP will alternately open and close the scanner pagemap window. Left buttoning on the item QUIT will close the input and output streams to the scanner, shutdown the RS232C port and close the scanner pagemap and control windows. The following icon will be displayed if you shrink the Microtek Scanner Control window.



The Microtek Pagemap window will close when you shrink the Microtek Scanner Control window and has to be explicitly opened when the Microtek Scanner Control window is expanded again. This is done by buttoning on PAGEMAP in the Microtek Command Menu window.

Within Interlisp you normally cannot create bitmaps larger than approximately 2.1 million pixels (about 1400 x 1400). The Microtek scanner software allows you to create bitmaps much larger than this but at the cost of using a correspondingly large amount of virtual memory. If you are near your maximum vmemsize, as determined by comparing (VMEMSIZE) to (VOLUMESIZE 'volumename), there is a good chance you could crash your system if you create a very large bitmap...caveat emptor. In addition you will not be able to call the function EDITBM to edit bitmaps larger than 2.1 million pixels

The reduction % used to scan the original image is stored on the property list of the atom that the bitmap is bound to. It is saved as the property "Resolution" and is in %. This is used to determine the appropriate values that will make an image 1:1 when printed. If you attempt to print a bitmap to an Interpress printer that was not created by use of the Microtek scanner software you will be prompted to enter a scale explicitly. The following table indicates the 8044 laser printer scale used for scanned images and can be used as a guide when attempting to print bitmaps not created by the Microtek software.

<u>REDUCTION (%)</u>	<u>RESOLUTION (DPI)</u>	<u>SCALE</u>
0	300	.240
5	285	.252
10	270	.266
15	255	.282
20	240	.300
25	225	.320
33	200	.360
35	195	.369
40	180	.400
45	165	.439
50	150	.480
55	135	.533
60	120	.600
67	100	.720
70	90	.800
75	75	.960

Further information about the Microtek scanner can be obtained from:

Microtek Lab Inc  
16901 South Western Avenue  
Gardena, California 90247  
Tel: 213-321-2121, 800-654-4160

---

---

## MONITOR

---

---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: COURIERSERVE, BITMAPFNS

MONITOR is a remote screen monitor which shows a scaled down version of the entire remote screen and a small section at full size which can be moved around.

The module contains the code for the client and the server and must be loaded on both. The program supports multiple instances of the tool, even at different scale factors, and works correctly between machines with different size displays.

The lower, full screen window is mouse sensitive. Pressing the left button in the window updates the upper, closeup window to contain the portion of the remote screen indicated by the cursor. Pressing the middle button in the full screen window causes the compressed image of the remote screen in the lower window to be updated.

(MONITOR *HOST* [*SCALE*])

[Function]

Opens a remote screen monitor onto *HOST*, where *HOST* is any specification that COURIER.OPEN accepts. *SCALE* is optional and determines the amount of compression of the remote screen bitmap as well as the amount of area covered by the closeup.

The useful range of scale factors is from 2 to about 8; a scale factor of *N* will compress the remote screen by  $1/N$  in width and height and the closeup will cover  $1/N^2$  of the area of the remote screen.

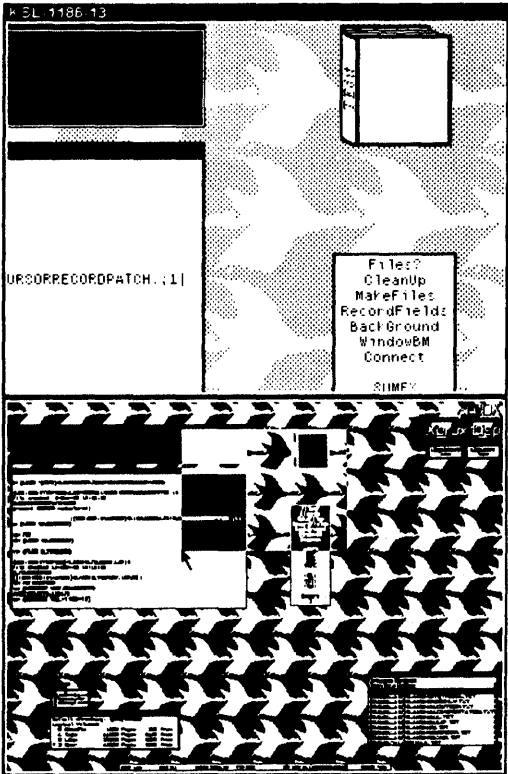
MONITOR.SCALE = 3

[Variable]

If not specified, the *SCALE* argument to MONITOR defaults to the value of MONITOR.SCALE.

### KNOWN PROBLEMS

- The Courier program number that the MONITOR Courier program uses is unregistered.
- The monitor does not yet correct for VIDEOCOLOR (which affects both the client and server).





---



---

## NEATICONS

---



---

By: Peter Schachte (quintus!pds@Sun.com)

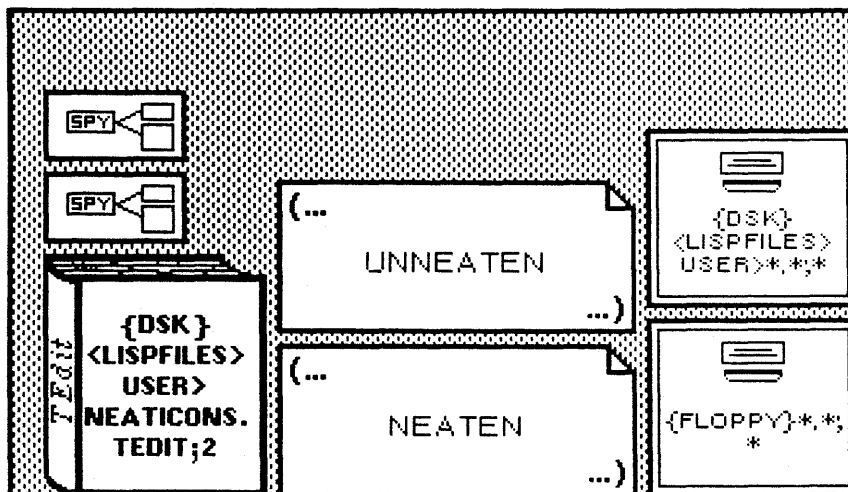
### INTRODUCTION

If you like to keep your icons neatly arranged on your screen, NEATICONS is for you. After this package is loaded, whenever an icon is created by shrinking a window, that icon will be "neat." But what *is* a neat icon? A neat icon is one that is lined up with another icon or window, or the edge of the screen. The easiest way to see this is to load the package, shrink a few windows (creating a snapshot and shrinking it is easy), and move them around. When a neat icon is moved near another icon or window or the edge of the screen, it is "grabbed" and moved neatly near it.

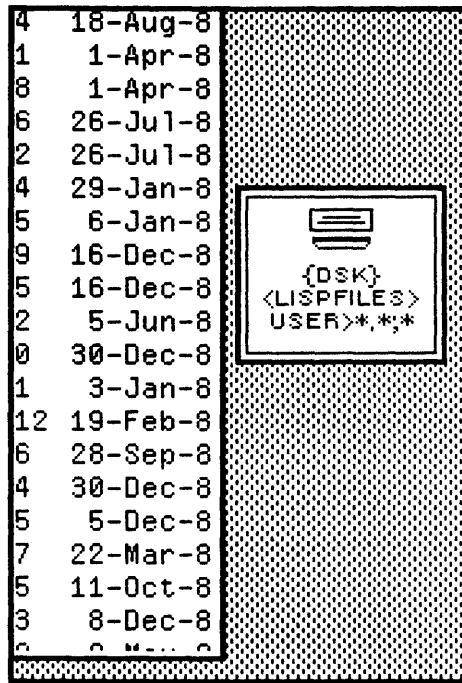
Neat icons line themselves up in a variety of ways. They will flush themselves with the edge of the screen. They will move themselves a fixed number of pixels from the edge of another window. Or they will align one of their edges with the corresponding edge of another window. When you move a neat icon, it will try to find a "neat" position near where you placed it, and place the window there instead. It may find a nearby position that is horizontally neat but not vertically, or vice versa. In any case, it will move the window into the nearest neat position it can find, or leave it where you put it if it can't find any nearby neat places.

### EXAMPLES

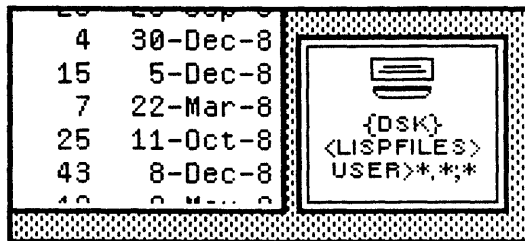
Here are a few examples of how your icons will be arranged. A typical cluster of neat icons:



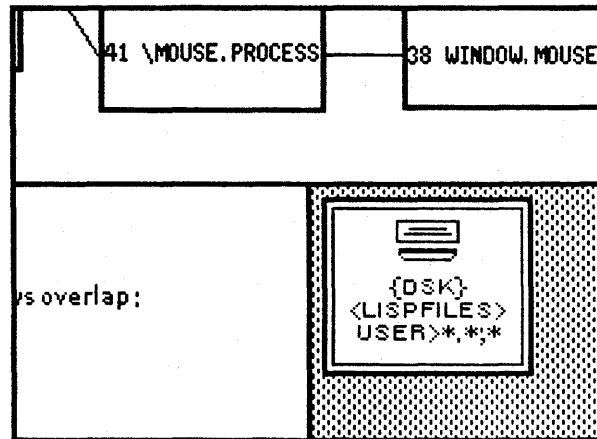
Neat icons can align themselves with the side of a window:



But more typically, they will align themselves with the corner of a window:



And they occasionally align themselves near the corner where two windows overlap:



#### DETAILS

You can just load this module and forget about it, and it will behave as advertised. It does have two user-settable parameters, and two user-callable functions, however. These are documented below.

Please note that the NEATICONS module is contained entirely in the NEATICONS package. This package exports the following symbols.

#### How near is near?

NEATICONS:DEFAULT-TOLERANCE [Variable]

This global parameter determines how many pixels vertically and horizontally an icon will be moved in order to make it neat. This defaults to 100.

#### Spacing between icons

NEATICONS:DEFAULT-SPACING [Variable]

This global parameter determines how many pixels apart neat icons will be placed. The default is 5.

#### Making a window neat

Loading the NEATICONS module causes SHRINKW to be advised, so every time a window is shrunk, the icon is made neat. So icons created before you load NEATICONS will not be neat, but they can be made neat by expanding and then re-shrinking them.

But suppose you want to make a regular window neat?

(NEATICONS:NEATEN &OPTIONAL WINDOW) [Function]

makes WINDOW neat. WINDOW defaults to (WHICHW), so you can point at a window with the mouse and type (NEATICONS:NEATEN).

**Making a window sloppy**

(NEATICONS:UNNEATEN &amp;OPTIONAL WINDOW)

[Function]

makes WINDOW no longer neat. It behaves just like a normal, sloppy, vanilla window. WINDOW defaults to (WHICHW).

**Other MOVEFNs**

The NEATICONS module uses each window's MOVEFN prop. If you wish to have another MOVEFN on a neat window, you can. If a window has MOVEFN when it is NEATICONS:NEATENed, it will be preserved. If you wish to add a MOVEFN to an existing neat window, you should put it on the window's

NEATICONS:USERMOVEFN

[Window Prop]

prop. This prop should hold a list of functions. When a neat window is moved, first it finds the nearest neat place. Then the first function on the window's NEATICONS:USERMOVEFN prop is called with the neat position as argument. If this function returns IL:DON'T, the window won't be moved. If it returns NIL, the position argument it was passed is passed to the second function on the list. If it returns a position, this position is passed to the second function on the list. The result of each function on the list is treated similarly, until all the functions have been called. The latest position is used as the position to move the window to.

---

---

## NOTEPAD

---

---

By: D. Austin Henderson, Jr. (AHenderson.pa@Xerox)

Compiled for Medley by Larry Masinter (Masinter.PA@Xerox.COM)

### Auxiliary file: NOTEPAD-CORESTYLES

**NOTEPAD** is a module for creating NOTEPAD windows - windows in which one can do artwork at the bitmap level. The ideas in this module come pretty directly from other people's work, both inside and outside Xerox, including Markup, Draw and Smalltalk. Notepad as it stands is the product of about a man-week of work, using standard Interlisp-D as released and the EDITBITMAP module of bitmap manipulation functions. It provides a nearly unusable interface to some distinctly interesting functionality. Comments and suggestions are welcomed.

### NOTEPAD (BITMAP COLORFLG)

Creates a NOTEPAD window. If BITMAP is NIL, then you are prompted for region for the window. Otherwise, a region is defined from the size of BITMAP, and you are prompted to move it to a desired position. If COLORFLG is non-NIL, the NOTEPAD window will be used only for control (for menu's, etc.), and all the painting will take place on the color screen.

There are two menus: one in the title of the window, and one in the window proper. Both are invoked by buttoning with either left or middle buttons.

#### Title menu

This menu gives access to commands for manipulating the window as a whole:

- New Notepad
- Copy Notepad
- Save as a bitmap.

#### Window menu

This menu has two columns of commands: those in the left column of the menu are for painting/erasing material into/from the bitmap (if this menu is invoked with the left button, painting is implied; if with the middle button, erasing); those in the right column of the menu are for changing the style in which the operations work.

#### Painting/Erasing

You can paint/erase using trajectories, or using (or editing) single objects. The commands in the left column of the menu are divided into two sets which reflect rthis division.

### Trajectories

Sketch: follows the mouse to define a trajectory of points at which to sketch.

Line: prompts for endpoints (fix and rubberband) and uses the points on the line as a trajectory.

Circle: prompts for center and a point on the circumference, and uses the points on the line as a trajectory.

Ellipse: prompts for center, end of semi-major axis and end of semi-minor axis, and uses the points on the line as a trajectory.

Open curve: prompts for first point and one or more subsequent points. You indicate that you are finished by depressing the left shift key on the last point defining the curve. A smooth curve is fitted through these points and used as a trajectory.

Closed curve: Like open curve, except that the fitted curve is closed, starting at the last point given, and proceeding to the first and then subsequent points.

### Objects/editing

Text: prompts for text, and permits positioning it with the mouse.

Area of the screen: Prompts for a (rectangular) region of the screen and places permits placing them where you want in the window.

Shade rectangle: prompts for a region, and paints/erases it with the current shade (a shade of black does complete paint and erase).

Fill: Prompts for a region within which to fill, and a point within the area to be filled; fills the area (not necessarily a rectangle, but defined by being closed rectilinearly) with the current shade.

Edit area: prompts for a region of the window and invokes the (Trillium) bitmap editor (standard Interlisp-D bitmap editor is the "hand-edit" choice on the submenu; others allow reflecting, rotating, shifting, inverting, putting on borders, etc.) on it. If the bitmap resulting from the edit is the same size as the original, it replaces the original region; if not, you are prompted for a place to put it.

### Style

Notepad operations (see above) are carried out in a style. The style at any given moment is given by a collection of characteristics. The current style can be saved (use the command SAVE.STYLE) under a name and restored (RESTORE.STYLE). Styles may be deleted (DELETE.STYLE) from the collection of saved styles. The style collection is currently part of notepad. Consequently, moving styles between loadups is not directly supported. (It is always possible to save it on a separate file. The styles are stored as the value of NOTEPAD.STYLES. It includes bitmaps, and must therefore be added as an UGLYVARS.)

The characteristics in the style are:

Brush: A bitmap which is either painted or erased at each point on or resulting from (see symmetry) a trajectory (see the operations paint, line, circle, ellipse). DEFINE.BRUSH prompts for a region which will then become the brush. EDIT.BRUSH permits editing of the brush bitmap (using the same editor as the operation EDIT.AREA - see above). BRUSH = COOKIE.CUT.WITH.MASK also defines a brush (see mask, below).

Use mask: An indication of whether or not to use the masking function (see mask, below) before painting/erasing. USE.MASK toggles this setting.

Mask: A bitmap which is used to clear out an area before the brush is used. The mask is erased if the brush is painting, and visa-versa. DEFINE.MASK prompts for a region which will then become the mask. EDIT.MASK permits editing of the mask bitmap (using the same editor as the operation EDIT.AREA - see above). MASK = OUTLINE.OF.BRUSH defines the mask to be the same size as the brush, with a pattern (of black) which is the filled-in outline of the brush. BRUSH = COOKIE.CUT.WITH.MASK allows you to move the mask over a section of the window and define the brush as the points so covered.

Use grid: An indication of whether or not to use the gridding function (see grid, below) while painting/erasing. USE.GRID toggles this setting.

Grid: An origin and the point (1, 1) of a grid to be used to "attract" the points used in following a trajectory. DEFINE.GRID prompts for the origin and (1, 1) point of the grid.

Use symmetry: An indication of what sort of symmetry function to use while painting/erasing. USE.SYMMETRY permits setting it as you choose. The choices are none, left/right, up/down, 4-fold (both left/right and up/down) and 8-fold (4-fold plus reflecting about the 45-degree diagonals). The brush/mask used when painting/erasing symmetrically can themselves be either identical to the brush/mask in use or symmetrically reflected. USE.SYMMETRIC.BRUSH/MASK toggles this setting.

Point of symmetry: The point with respect to which the symmetry functions are defined. POINT.OF.SYMMETRY prompts for this point.

Text font: The font in which text is printed. DEFINE.FONT permits choosing one of the fonts already loaded or OTHER (in which case you can type in the font description (family size face)).

Shade: The shade used for the rectangle and fill operations. EDIT.SHADE permits you to edit the shade (using the standard Interlisp-D shade editor).

---

---

**NSCOPYFILE**

---

---

By: Bill van Melle (vanMelle@Xerox.com)

The module NSCOPYFILE modifies COPYFILE so that if both the source and destination files are on NS file servers, the copying is done by an NSfiling-specific copy routine. This routine copies *all* attributes of the source, including non-standard ones, such as those used by Viewpoint. Thus, you can safely copy Viewpoint files from inside Lisp without losing information. In addition, if the copy is from an NS file server to itself, the copy is performed by the server itself, which is considerably faster than shipping the file over the Ethernet.

In addition, you can also copy entire directories, by specifying directory names as the source and destination, e.g.,

```
(COPYFILE "{FS:}<Carstairs>Lisp>" "{FS:}<Calvin>Lisp>Current>")
```

The destination directory must not already exist, since this operation creates an entirely new directory, whose contents are a copy of all the source directory's offspring, to all levels. If the destination directory happens to exist but has no children, it is considered vestigial and is quietly deleted first (Lisp usually suppresses such directories when performing a directory enumeration).

You can also use RENAMEFILE in the same manner to either rename a directory, or to move an entire directory and its descendents to a new node in the file server's hierarchy, or to a new server altogether. You must, of course, have access rights to delete the source directory *and* all its children, and the destination must be on an NS file server.

A word about protection: when a file is copied or moved, the new file is given "defaulted" access rights, i.e., its protection is set as specified by its new parent (sub)directory, just as if you had created the file afresh by any other means. Thus, if the original file happened to have its own explicit protection, that protection is ignored. When copying or moving an entire directory, only the top-level directory receives default protection, so if any individual descendent file had non-default protection, that protection is copied verbatim. This can lead to confusion—you may want to use the NSPROTECTION module to change the new directory's descendents to default protection. See the documentation of NSPROTECTION for more discussion about protection issues. Note that if a file/directory is renamed within the same parent directory, the operation is considered merely "changing the name", and its protection is left unchanged.

Note: If you are using the FILEWATCH module, be aware that files being copied between NS servers do not appear (because the files are not opened by the normal Lisp file system).



---



---

## NSDISPLAYSIZES

---



---

By: Bill van Melle (vanMelle.pa@Xerox.com)

The NS font families all have screen fonts that display at approximately their nominal point size. This means that there is a closer congruence between their appearance on the display and on hardcopy than there is for, say, the Press fonts. Unfortunately, this means that the NS display fonts are "too small"—a size that is quite readable on hardcopy can be uncomfortably small on the display. The module NSDISPLAYSIZES attempts to ameliorate this problem by fooling FONTCREATE into using bigger fonts for display without changing anyone's belief in the nominal size of the font.

Loading NSDISPLAYSIZES.LCOM modifies FONTCREATE's font file lookup procedure so that a request for NS display font of size  $n$  actually reads the font file for size  $n+2$ . For example, (FONTCREATE 'MODERN 10) will actually read the display font file belonging to Modern 12, but FONTCREATE will still believe the resulting font is Modern 10. Font sizes greater than 12 are not affected, on the grounds that those fonts are already big enough to read, and not all fonts are available in size  $n+2$  for large  $n$ ; hence, for example, Classic 12 and Classic 14 will end up using the same actual font for display. Also, since Terminal 14 does not yet (as of this printing) exist, Terminal 12 remains Terminal 12.

A font is considered an NS font if its name is a member of the list NSFONTFAMILIES, whose initial value is (CLASSIC MODERN TERMINAL OPTIMA TITAN).

Loading the module clears the internal font cache of all NS display fonts, so that subsequent calls to FONTCREATE will not erroneously return a font cached earlier under the default lookup procedure. Of course, if someone has already set some font variable to (FONTCREATE 'MODERN 10), that font descriptor will not be affected.

Note that this module has no effect on hardcopy—a font is always printed at the size you named it. And you can still use TEdit's Hardcopy display mode to see how a piece of text will be formatted on the printer.

If the VIRTUALKEYBOARDS module is present, then loading NSDISPLAYSIZES automatically edits its keyboard specifications so that it continues to use Classic 12 in its keyboard displays. Without this fix, the keyboard display routines will try to create Classic 12, which NSDISPLAYSIZES coerces to Classic 14, and as a result the keyboards will be poorly displayed and lack many characters (since Classic 14 is not as complete as 12). If you load VIRTUALKEYBOARDS *after* NSDISPLAYSIZES, you should call (VKBD.FIX.FONT) yourself to make the change.

Note that other modules can have similar problems if they have hardwired into them a specific size of NS font as being appropriate for their display configuration. For such modules to operate correctly in the presence of NSDISPLAYSIZES, they might want to be aware of the function used to coerce sizes:

(NSDISPLAYSIZE *FAMILY SIZE FACE EXTENSION*) [Function]

Returns a font size that (FONTCREATE *FAMILY SIZE FACE*) will use instead of *SIZE*. *EXTENSION* must be a member of DISPLAYFONTEXTENSIONS in order that we know we are doing this for the

display. Follows the rules described above. For example, (NSDISPLAYSIZE 'MODERN 12 'MRR 'DISPLAYFONT) returns 14. (NSDISPLAYSIZE 'GACHA 12 'MRR 'DISPLAYFONT) returns 12.

In the simplest case a module could just test for the existence of NSDISPLAYSIZE and choose one size or another. For example,

```
(SETQ MYFONT (FONTCREATE 'MODERN (if (GETD 'NSDISPLAYSIZE)
                                     then 10
                                     else 12)))
```

---

---

## NSPROTECTION

---

---

By: Bill van Melle (vanMelle@Xerox.com)

### INTRODUCTION

The module NSPROTECTION provides a tool that enables you to easily change the protection of files and directories on Xerox NS file servers.

To install the module, load the file NSPROTECTION.LCOM. Also, Your NS file server must be running Services release 10.0 or later.

### THE PROTECTION MECHANISM

An NS File Server maintains a protection for each file and (sub)directory on the server. In most cases, the protection is not specified explicitly, but rather is inherited from a file's parent directory, making it easy to maintain consistent protection over an entire branch of the file system hierarchy.

The protection is specified as a set of pairs <access rights, name>. The name can be the name of an individual user or a group. The name can also be a pattern of the restricted form \*:domain:organization, \*:\*:organization, or \*:\*:\*. The access rights granted to any particular user are the most general of those in the pairs that match the user's name (by exact match, pattern or membership).

The following five kinds of access rights are independently specified (the term "file" here can also denote a directory in the places where that makes sense):

- Read The user may read the file's content and attributes. In the case of a directory, the user may enumerate files in it.
- Write The user may change the file's content and attributes, and may delete the file. In the case of a directory, the user may change the protection of any of the directory's immediate children.
- Add (Applies only to directories) The user may create files in the directory (i.e., add children).
- Delete (Applies only to directories) The user may delete files from the directory (i.e., remove children).
- Owner The user may change the file's access list.

In the case of directories, it is also possible to independently specify the directory's own protection and the protection that its children inherit by default. In most cases, the latter simply defaults to the former, and it is usually best to keep it that way for simplicity. However, there might conceivably be cases where, for example, you would want a user to be able to read the files in a directory, but not be able to enumerate it, or vice-versa.

Note that there can be problems when giving a more lenient protection to a file or directory than to its parents, depending on what software is going to be used to gain access to the file. For example, if your default directory protection grants access only to you, and you want to allow a

user to read a particular file stored in your directory, then you can change the protection on just that file to allow Read access. However, the user will have to know the exact name of the file in order to read it, since she won't be able to enumerate the directory to search for the file. Specifying the exact file name works fine from Lisp, but other software that gets to a file by starting at the top and working its way down through the hierarchy would be unable to get to the file.

## USER INTERFACE

To use the tool, select "NS Protection" from the background menu (if your menu has a "System" item, it's a subitem underneath it), or call the function (NSPROTECTION). You are prompted for a place to position the tool's window. Be sure to leave space below the window for the protection information that will follow.

NS File Protection Tool			
<b>Show</b>	<b>New Entry</b>	<b>Apply</b>	<b>Set to Default</b>
<b>Type:</b> Principal		<b>Check:</b> New Names Only	
<b>Host:</b>			
<b>Dir/File:</b>			

The tool window has four command buttons across the top, two switches labeled **Type** and **Check**, and two fill-in fields for the host and file name. Holding a mouse button down over any of these items for a couple of seconds will display a help message in the prompt window.

To view or change the protection of a file or directory, first fill in the **Host** and **Dir/File** fields. You can edit these fields by clicking with the mouse anywhere inside the existing text (if any), or by clicking with the LEFT button on the boldface label. If you click with RIGHT on the label, then any existing text is first erased. Typing the Next or Return key moves to the next field. [See the FreeMenu documentation for more information about text editing.]

You can either enter the host and directory separately, e.g.,

```
Host: Phylex
Dir/File: Carstairs>Lisp
```

or enter a file name in the usual Lisp syntax in the **Dir/File** field, e.g.,

```
Host:
Dir/File: {Phylex:}<Carstairs>Lisp>
```

This latter form is intended to make it easy to copy-select the name of the directory or file from another source, such as a FileBrowser window; the host in the full name overrides any name in the **Host** line.

To see the protection of a file or directory, click on the command **Show**. The protection is displayed as a series of editable one-line windows beneath the main window. In each line is a set of access rights and a Clearinghouse name or pattern to which those rights are granted; for example,

NS File Protection Tool				
<b>Show</b>	<b>New Entry</b>	<b>Apply</b>	<b>Set to Default</b>	
<b>Type:</b> Principal		<b>Check:</b> New Names Only		
<b>Host:</b> Phylex;Research;ACME				
<b>Dir/File:</b> <Carstairs>Lisp>				
<b>Read</b>	<b>Wrt</b>	<b>Add</b>	<b>Del</b>	<b>Own All</b>
<b>to:</b> Jonathan Q. Carstairs;Research;ACME				
<b>Read</b>	<b>Wrt</b>	<b>Add</b>	<b>Del</b>	<b>Own All</b>
<b>to:</b> LispDevelopers;Research;ACME				
<b>Read</b>	<b>Wrt</b>	<b>Add</b>	<b>Del</b>	<b>Own All</b>
<b>to:</b> *:Research;ACME				

The highlighted buttons indicate which of the five access rights (Read, Write, Add, Delete, Owner) are granted to the name on the right. If the displayed protection was inherited from its parent subdirectory, rather than having been explicitly set, this fact is noted in the prompt window.

To change the protection of a file or directory, set up the protection entries as desired, then click on the command **Apply**. The usual procedure is to use the **Show** command to see the current protection, then edit one or more entries. Clicking on one of the first five buttons toggles it; clicking on **All** either sets all five (if **All** was previously unhighlighted) or clears all five. In addition, setting either **Write** or **Add** also sets **Read**, since they are of little use without read access (you can, however, clear **Read** if you really meant it). The name following **to** is edited in the same manner as the **Host** and **Dir/File** items above. As with most other places in the system, the name you type can omit the domain and organization, in which case the tool will fill in the local defaults; you can also use nicknames, which will be replaced by the Clearinghouse full names (assuming checking is on).

To add an additional entry, click on the command **New Entry**. This adds a new line to the existing set of protection entries, which you can edit as appropriate. To remove a set of access rights completely for an existing name, either clear all five access buttons (most easily done by clicking once or twice on **All**), or clear the name from the **to** field (by clicking on it with the RIGHT mouse button). Any such cleared lines will be removed by the **Apply** command.

You can also change the protection of a file back to "default" by clicking on the command **Set to Default**. Following this command, the protection of the specified file is inherited from its parent directory. This is usually the best way to "undo" a changed protection, because then any changes to the protection of its parent, or parent's parent, etc., will have the expected effect on all its children.

For the **Apply** and **Set to Default** commands, you may also specify a group of files, rather than a single file, by giving a file pattern—a name with asterisks serving as wild cards to match zero or more characters. Any pattern acceptable to the File Browser can be used. The tool enumerates the specified set of files and applies the specified protection to each. The enumeration is made to all levels (infinite depth), so affects files both in the immediate directory and also in its subdirectories, and subdirectories of those, etc. The enumeration does not, however, include the top-level subdirectory itself; e.g., "<Carstairs>Lisp>\*" matches all files (including subdirectories) anywhere in the directory <Carstairs>Lisp>, but does not include <Carstairs>Lisp> itself.

Note that applying a protection to a directory is different from applying the same protection to the files in it, because of defaulting. If you apply a protection to <Carstairs>Lisp>\*, it changes the protection of every file currently in the directory, but any new files added after the change still inherit the protection of the directory <Carstairs>Lisp>. On the other hand, applying a protection to the directory <Carstairs>Lisp> itself affects all current and future files in the

directory, *except* any files that already have an explicit protection currently set. To reduce confusion, it is thus preferable to apply protections to subdirectories, rather than individual files, if you want to control a whole group of files. If you have a subdirectory containing files of miscellaneous protection that you would like to make uniform, the best procedure is to set the desired protection on the subdirectory itself, and then use the **Set to Default** command with a pattern (e.g., `<Carstairs>Lisp>*`) to reset all the individual files to defaulted.

The **Apply** command looks up in the Clearinghouse each of the names in the individual protection entries to make sure that they are valid, and replaces aliases (nicknames) with the canonical names. It then tells the file server to change the protection as indicated. The extent to which the **Apply** command checks names is controlled by the **Check** item in the second line of the tool window. It has four possible settings:

- New Names Only** This is the default setting. The tool checks any names that you have entered or changed, but assumes that names returned by the **Show** command were correct.
- All Names** The tool checks all names, regardless of source. You might want to do this to convert an existing protection entry into canonical form, or check that all the names are still valid.
- Never** The tool never checks names; it assumes you meant exactly what you typed. You might want this setting, for example, if one of the names you are entering is registered only in a distant Clearinghouse not currently accessible.
- I really mean it** Not only does the tool not check the names, it also doesn't balk if you tell it to take certain unlikely actions, such as changing a top-level directory to default protection, setting a completely null protection, or setting a protection in which nobody has Owner rights (which means the protection can only be changed by someone with Write access to the parent, if any). This setting is "one-shot"—it reverts to "New Names Only" after you issue the next command.

The **Type** item in the second line of the tool window controls which of a directory's two protection attributes is displayed or set. The initial setting is "Principal" and is the one that should normally be used (it coincides with the Lisp file attribute PROTECTION, or "Access List" in NS Filing parlance). The other setting is "Children Only". When the protection type is set this way, the tool deals with the protection that is inherited by default by the directory's children, the attribute called "Default Access List" in NS Filing parlance. Ordinarily, this attribute is defaulted, in which case the directory's principal protection is also used as its children's default protection. Using the **Apply** command changes the Default Access List to the value you specify; using the **Set to Default** command changes it back to defaulted. The **Show** command displays the directory's Default Access List if it has one; otherwise, it displays the principal protection and notes this fact in the prompt window.

The **Type** item is irrelevant for non-directory files (and, in fact, the tool sets it back to "Principal" if it has been changed). When the file is a pattern, the tool always sets the Principal protection; in the case of any subdirectories matching the pattern, it sets the Principal protection to that specified in the window and the Default Access List to "default".

As an additional convenience feature, when you request to **Show** the "Principal" protection of a top-level directory, the tool also displays in the prompt window the directory's current page usage

and allocation.

---

---

**PAGEHOLD**

---

---

By: Jon L White

Currently maintained by: Bill van Melle (vanMelle.pa@Xerox.com)

**INTRODUCTION**

Loading PAGEHOLD.LCOM redefines the function PAGEFULLFN to alter the behavior that occurs when a tty window fills. Rather than inverting the window and waiting indefinitely for type-in, the PAGEHOLD module indicates the hold by an independent notification, and waits for only a specified interval before continuing. Thus, filling the window is no longer a cause for a program to hang indefinitely.

The default behavior of the PAGEHOLD module is to raise a "button" at the corner of the tty window flashing a message, alternating between



-- SHIFT to hold typeout --

and



-- Release SHIFT for more --

indicating that output to the window is being held. While in this state, you can release the hold by any of the following means:

Typing any character (of course, the window must own the tty process). This is the same as the old behavior;

Depressing the CTRL key;

Depressing and releasing either SHIFT key;

Clicking with LEFT on the button that announces the hold (clicking instead with MIDDLE gets a menu of options);

Waiting until the timeout has passed (initially, 20 seconds).

When you depress one of the SHIFT keys, the button stops flashing. Output will continue to be held indefinitely as long as one of the SHIFT keys is depressed, even if the timeout passes. If while holding down SHIFT, you depress the CTRL key for a second or so, the button will start flashing again; you may now release CTRL and then SHIFT, and the hold will be maintained without your needing to hold down SHIFT. You can release the hold by any of the means listed above.

If the CTRL key is down when a window fills, output is not held at all. Depressing the CTRL key immediately releases any hold in progress.

The remainder of this document describes ways of tailoring the behavior further.



### Controlling the timeout

One of the primary motivations for the PAGEHOLD module is so that printout to a TTY window does not hang indefinitely when one "page" has filled up. The default release time is in the global variable `PAGE.WAIT.SECONDS`, which comes initialized to 20 seconds; a value of 0 causes immediate release (unless a SHIFT key is already depressed). If a window being held has a `PAGE.WAIT.SECONDS` property, then that value is used instead of the global default.

However, if `PAGE.WAIT.SECONDS` is set to `STOP`, then the hold will not be released by any automatic timeout, nor will it be sensitive to the SHIFT or CTRL key actions. This mode most closely approximates the current Lisp design, except that a pop-up button signals the hold rather than a video inversion (mousing the button will, nevertheless, still effect a release). The message

"Scrolling Stopped"

appears in the button rather than one of the several "holding" messages.

### The Pop-up "Buttons"

A secondary motivation for this facility is to have a pop-up "button" that interactively signals the user of a holding condition on a particular window without obscuring the window's contents, as video inversion does. In addition, the button permits the selective release of a particular window by mousing the button (note that holding down SHIFT, on the other hand, would affect *all* windows currently being held). There are three styles of buttons—`WINKING`, `FLASHING`, and `NIL`—and the selection is determined by the value of the global variable `PAGE.WAIT.ACTIVITY`, which comes initialized to `WINKING`. If a window has a `PAGE.WAIT.ACTIVITY` property, then that value is used instead of the global default, thus allowing different types of buttons on different windows.

A `WINKING` button is a fairly hefty pad—approximately 1/2" by 2 1/2"—which pops up just over the right side of the window's title bar; it will alternately print and clear two short holding messages: one in the upper half of the "button" and one in the lower half. A `FLASHING` button is about the same width, but half the height, and will alternately print the two holding messages. A `NIL` button merely shows the message "Release SHIFT for more".

LEFT-mousing any button causes an immediate release of the hold; MIDDLE-mousing the button brings up a menu offering several options. One of these is "Release this hold!", same as using LEFT. Other menu options permit conversion of the hold to indefinite "hold" or to `STOP` mode; additionally, five options are offered for setting the window's specific `PAGE.WAIT.SECONDS` property.

The `WINKING` button has a different pattern of activity when the hold is placed into indefinite hold mode, but the other button styles do not visibly distinguish this state. If there isn't room to place the button down over the right side of the title bar (because, for example, the window is too close to the screen top), then it will be placed over another corner of the window.

### Keyboard Input and Typeahead

Consistent with Lisp's current action, there will be no holding on a window which is its process's `TtyDisplayStream` and for which there is typeahead in that process's TTY input buffer. This action can be overridden by setting `PAGE.WAIT.IGNORETYPEAHEAD` to a non-NIL value: typeahead does not inhibit the hold, character input does not release the hold, and no input is ever discarded (note that depressing the SHIFT and/or the CTRL keys does not generate character input). This feature is intended for those who dislike not knowing whether a keystroke will be consumed by

the PAGEFULLFN—under the default behavior, if the TTY input buffer is empty, then the first character you type will either (a) release a hold already in progress and be discarded, or (b) prevent subsequent holds and be retained, all depending on when exactly you type the character.

---



---

## Piece-Menus

---



---

By: D. Austin Henderson, Jr. (AHenderson.pa@Xerox.COM)

### INTRODUCTION

This module provides two solutions to the problem of menus with too many items. Which is useful will depend on the inherent structure of the items. 1) **CHUNK-MENUS**: For use in which the items have the simple structure of one long list: Break the items up into chunks following the ordering of the items in the list, and then provide a menu which presents one of those pieces and a way of getting other menus containing the other pieces. 2) **KEYWORD-MENUS**: For use when it is possible to associate keywords with the items: Break the items up into the sets which share the same keyword, and then provide a menu which presents one of those chunks and a way of getting other menus with other keywords. As with standard Interlisp-D menus, these specialized menus are created by a "create" function (cf. (CREATE MENU)), and are used with a single "invoke" function (cf. (MENU menu)). The items and other information are as with standard menus, except that the other information is presented to the create function in the form of a Plist.

### CHUNK MENU

A Chunk Menu appears as a single menu, but is really a data structure encompassing a number of pieces, each represented by a menu, between which the operator can move to find the item desired. Each piece is in three parts: a set of "required items" which will appear in all chunks, a set of indicators for all the chunks, and the items in this piece. Each chunk has up to 30 items in it (it can be varied using the `CHUNK.COUNT` property). If there are fewer than one chunk's worth of items, the **CHUNK-MENU** behaves just like a standard menu.

(`CHUNK.MENU.CREATE ITEMS PROPERTIES REQUIRED.ITEMS`) [Function]

Creates and returns a Chunk Menu, a data structure of menus each containing some of the (presumably large number of) items.. All items are as with standard menus. The properties understood are: `CHUNK.COUNT` (see above), `TITLE`, `CENTERFLG`, `MENUFONT`, `ITEMWIDTH`, `ITEMHEIGHT`, `MENUBORDERSIZE`, and `MENULAYOUTSIZE`. The actual menus are created only when needed.

(`CHUNK.MENU.INVOKE CHUNK.MENU POSITION`) [Function]

Carries out the interaction with the user to select an item from a Chunk Menu. If `POSITION` is non-NIL, the menu will appear at that position, otherwise it will appear under the mouse. All interactions are as with the standard menus. Returns the value produced when a selection is made. Selecting outside any of the menus appearing in the interaction cancels the interaction and returns NIL.

### KEYWORD MENU

A keyword menu appears as a single menu, but is really a data structure encompassing a number of menus which the operator can move between to find the item desired. Each piece is in two parts: the set of keywords for all the pieces, and the items in this piece (having this keyword). Each piece is a Chunk Menu, so that if a particular keyword has many items, its piece is itself broken into pieces. The items in a keyword menu is computed from a list of objects, the things which

presumably are being selected among. A function is provided for computing the keywords of each object, and another for computing an item from an object.

**(KEYWORD.MENU.CREATE *OBJECTS KEYWORDFN PROPERTIES ITEMFN*)** [Function]

Creates and returns a Keyword Menu, a data structure of pieces associated with the keywords of the *OBJECTS* as determined by *KEYWORDFN*. Each piece of the Keyword Menu is a Chunk Menu and contains the items for the objects with the associated keyword. The item is computed from the object by applying the *ITEMFN* to that object; *ITEMFN* should return an item appropriate for standard menus. The menus are determined by the values on the PList *PROPERTIES*. The properties understood are: *CHUNK.COUNT* (see Chunk Menus), *TITLE*, *CENTERFLG*, *MENUFONT*, *ITEMWIDTH*, *ITEMHEIGHT*, *MENUBORDERSIZE*, and *MENULAYOUTSIZE*. The actual menus are created only when needed.

**(KEYWORD.MENU.INVOKE *KEYWORD.MENU POSITION* )** [Function]

Carries out the interaction with the user to select an item from a Keyword Menu. If *POSITION* is non-NIL, the menu will appear at that position, otherwise it will appear under the mouse. All interactions are as with Chunk Menus, which is just that for standard menus for small numbers of items. Returns the value produced by the items when a selection is made. Selecting outside any of the menus appearing in the interaction cancels the interaction and returns NIL.

---

---

## PLOT

---

---

By: Jan Pedersen (pedersen.PA @ Xerox.com)

Uses: TWODGRAPHICS and PLOTOBJECTS

PLOT is a module designed to assist in the production of analytic graphics. PLOT provides automatic scaling, labeling, incremental modification, generalized selection, and a collection of standard graphics primitives which may be combined to produce interactive plots of great diversity.

PLOT is to some degree object-oriented. The primitive components of a plot are plot objects (e.g. points, lines, etc.). A plot manager maintains a display list of plot objects which are individually responsible for displaying themselves, highlighting themselves, etc. The user constructs a plot incrementally, adding plot objects, while the plot manager handles details such as the appropriate scale for the plot. Each plot object is active, in the sense that it is selectable and may have a menu associated with it. In addition, the plot manager may be directed to modify the appearance of the entire plot through a command menu.

The module is open, in the sense that most default behaviors may be overridden by the user, although it is hoped that the defaults will be sufficient for most applications. A functional interface is provided for programmatic access to all of PLOT's facilities.

The plot manager is abstracted as a datatype of type PLOT, along with a collection of functions which operate on PLOT's. Functions are provided to create PLOT's, manipulate their display lists, and modify default menus. Plot objects are abstracted as instances of datatype PLOTOBJECT. A set of default plot objects are provided, along with a mechanism of defining new plot objects.

Plots exist independently of their representation on the screen. Indeed, it is intended that plots may be displayed on ANY imagestream. However, the most common usage is to display a plot in a window, and a PLOT does have an associated WINDOW which may be opened, closed, etc.

Plots may be hard copied, made into image objects, and dumped to file.

The lispuser's module PLOTEXAMPLES contains a few examples of how PLOT may be used to create high level plotting facilities.

### BASIC OPERATION

A plot is abstracted as an instance of datatype PLOT which includes a display list, a property list, and an associated window, among other things. PLOT's may be create via the function CREATEPLOT.

(CREATEPLOT *openflg region title border*)

[Function]

Returns a PLOT. If *openflg* is T then the PLOT's associated window is opened with an empty plot. The other arguments are treated as in CREATEW.

An empty plot is initialized to have a world coordinate system extending from 0.0 to 1.0 on either axis, with no labels or tic marks displayed. As objects are added to the plot, the world coordinate system is grown to accommodate the new objects.

A PLOT has an associated window, which is closed by default. The window is used as the primary display device and may be manipulated with the following functions.

(OPENPLOTWINDOW *plot*) [Function]

Opens the plot's associated window.

Returns the associated window.

(CLOSEPLOTWINDOW *plot*) [Function]

Closes the plot's associated window.

(REDRAWPLOTWINDOW *plot*) [Function]

Redraws, by running down the current display list, the contents of the associated window. Opens the window if it is closed.

(GETPLOTWINDOW *plot*) [Function]

Returns the window associated with plot.

(WHICHPLOT *x y*) [Function]

Returns the PLOT associated with the window (or icon) at position (*x . y*), or at the current cursor position if *x* and *y* are defaulted. *x* may be a WINDOW, in which case the associated PLOT is returned.

A plot object is abstracted as an instance of datatype PLOTOBJECT. A point plot object is an instance of PLOTOBJECT whose data component describes a point. That is, a point plot object is a subtype of PLOTOBJECT; all plot objects satisfy (type? PLOTOBJECT FOO), but only a point plot object satisfies in addition (PLOTOBJECTSUBTYPE? POINT FOO). A collect of standard plot objects has been implemented, including point, curve, polygon, line, and filled rectangle plot objects. The module is designed so that new objects may be defined at any time, but that mechanism is described in a separate document.

PLOTOBJECT's may be added to or deleted from a PLOT. The following functions provide an add facility for the standard objects.

(PLOTPOINT *plot position label symbol menu nodrawflg*) [Function]

Only the plot and position arguments are required. Position is a POSITION in world coordinates. Label is an expression which will be PRIN1 'ed whenever a label is required (typically an atom or a string). Symbol is a BITMAP which will be plotted centered at position. The litatoms CROSS, CIRCLE, STAR are bound to convenient BITMAPS. Symbol defaults to STAR. Menu is either a MENU, a litatom, in which case a MENU of that name must be cached on plot (more about this later), or an item list which may be coerced into a MENU.

If nodrawflg is non-NIL then a point object will be added to the display list of plot, but the associated window will not be updated. If Nodrawflg is NIL, and the plot's associated window is not open, it will be opened.

Returns a POINT PLOTOBJECT.

(PLOTPOINTS *plot positions labels symbol menu nodrawflg*) [Function]

As above except that positions is a list of POSITIONS and labels may also be a list. Reasonable things happen if positions and labels are of unequal length.

Returns a list of POINT PLOT OBJECT's.

(PLOT CURVE *plot positions label style menu nodrawflg*) [Function]

The list of POSITION's defines a piecewise linear curve. Style may be an integer which specifies the line width (in pixels) or a list of (linewidth *dashing color*), any of which may be NIL; defaults to one. For convenience the atoms DOT, DASH and DOTDASH have been bound to a few *dashing patterns*.

Returns a CURVE PLOT OBJECT.

(PLOT POLYGON *plot positions label style menu nodrawflg*) [Function]

As in PLOT CURVE, although a polygon is a closed figure

Returns a POLYGON PLOT OBJECT.

(PLOT TEXT *plot position text label font menu nodrawflg*) [Function]

Text should be a STRING to be printed at position.

Returns a TEXT PLOT OBJECT.

(PLOT FILLED RECTANGLE *plot left bottom width height label texture borderwidth menu nodrawflg*) [Function]

Texture must be TEXTURE. SHADE1, ..., SHADE8 are bound to some convenient textures. Defaults to SHADE3.

Returns a FILLED RECTANGLE PLOT OBJECT.

The following two functions add analytic plot objects to the display list of a PLOT. Analytic objects differ from points, curves, etc. by having infinite extents; their appearance on a plot depends on the current world coordinate scale, but adding an analytic object to a plot will not effect the current scale.

(PLOT LINE *plot slope constant label style menu nodrawflg*) [Function]

Slope and constant define an analytic line,  $y = \text{slope} * x + \text{constant}$ . If slope is NIL, it is taken to be infinite; i.e. the line is vertical.

Returns a LINE PLOT OBJECT.

(PLOT GRAPH *plot graphfn nsamples label style menu nodrawflg*) [Function]

Graphfn should be a function of one variable which defines a graph (or the graph of a function) to be drawn on plot. Nsamples is the number of equispaced points along the x-axis of plot at which graphfn is to be sampled when drawn; defaults to 100.

Returns a GRAPH PLOT OBJECT.

Complex objects may be built up from the preceding primitives by defining a compound plot object, which is simply a collection of other plot objects, including other compound objects.

(PLOT COMPOUND *plot component1 ... componentn typename label menu nodrawflg*) [NoSpread Function]

A compound plot object is specified by listing its components. In addition, a compound plot object may have its own menu and label. The typename field is supplied to allow different compound

objects to be differentiated. Drawing a compound object amounts to drawing its components recursively. In general, operations on compound objects are applied recursively.

Components 1 through n are plot objects. Typename is required and serves to tag this compound object, and is accessible via the function COMPOUNDSUBTYPE. Label and menu are as in other plot objects.

Returns a COMPOUND PLOBJECT.

All plot objects may be created independently of the previous functions. This is useful if it is desired to create a plot object without entering it on a PLOT's display list. The following functions create and return the standard plot objects.

(CREATEPOINT *position label symbol menu*) [Function]

Returns a POINT PLOBJECT.

(CREATECURVE *positions label style menu*) [Function]

Returns a CURVE PLOBJECT.

(CREATEPOLYGON *positions label style menu*) [Function]

Returns a POLYGON PLOBJECT.

(CREATETEXT *position text label font menu*) [Function]

Returns a TEXT PLOBJECT.

(CREATEFILLEDRECTANGLE *left bottom width height label texture style menu*) [Function]

Returns a FILLEDRECTANGLE PLOBJECT.

(CREATELINE *slope constant label style menu*) [Function]

Returns a LINE PLOBJECT.

(CREATGRAPH *graphfn nsamples label style menu*) [Function]

Returns a GRAPH PLOBJECT.

(CREATECOMPOUND *compoundtype components label menu*) [Function]

Components must be a list of PLOBJECT's.

Returns a COMPOUND PLOBJECT.

Each PLOT has a display list which is nothing more than a list of plot objects. The display list may be manipulated directly via the following functions.

(ADDPLOBJECT *plotobject plot nodrawflg*) [Function]

Interns plotobject on the display list of plot, and updates the associated window. The update is suppressed if nodrawflg is non NIL.

One might think of PLOTPOINT as being equivalent to:

(ADDPLOBJECT (CREATEPOINT *position ....*) *plot nodrawflg*)

Interns plotobject on the display list of plot, and updates the associated window. The update is suppressed if nodrawflg is non NIL.



Returns plotobject.

(DELETEPLOT OBJECT *plotobject plot nodrawflg nosaveflg*) [Function]

Deletes plotobject from the display list of plot, and updates the associated window accordingly. The update is suppressed if nodrawflg is T. If nosaveflg is T, then the deleted object will not be saved for possible later undeletion.

Returns plotobject if it was deleted from the display list, else NIL.

A PLOT has collection of properties, some of which are maintained by the plot manager, and others which may be used to cache arbitrary user data. All plot properties are accessed via the function PLOTPROP.

(PLOTPROP *plot prop newvalue*) [NoSpread Function]

If newvalue is absent then the current value of prop is returned. If newvalue is supplied (even if it is NIL) then the value of prop is set and the old value returned. The distinguished prop's PLOT OBJECTS, PLOTSCALE, SELECTEDOBJECT, PLOTWINDOW, PLOTWINDOWVIEWPORT, PLOT PROMPTWINDOW, and PLOTSAVELIST refer system maintained properties plot, and should be treated as read only. Compiles open in some cases.

For example, The display list of plot may be accessed by the expression.

(PLOTPROP plot 'PLOT OBJECTS)

For convenience in manipulating the property list of a PLOT, the following functions are provided.

(PLOTADDPROP *plot prop itemtoadd firstflg*) [Function]

If the value of prop is a list then itemtoadd is added to the end of the list. If the value of prop is NIL, it is set to (LIST itemtoadd). Firstflg indicates that the new item is to be the first in the list rather than the last. Works only for user defined properties.

Returns the new value.

(PLOTDELPROP *plot prop itemtodelete*) [Function]

If itemtodelete is a member (MEMB) of the prop value, it is deleted. Works only for user defined properties.

Returns NIL if nothing was deleted, else the new value of prop.

(PLOTREMPROP *plot prop*) [Function]

Destructively removes prop from property list of plot. Works only for user defined properties.

Each plot object also has a property list. As with PLOT's, some of the properties are maintained by the system, but the rest may be used to store arbitrary data objects. The property list of a plot object is accessed through the function PLOT OBJECTPROP.

(PLOT OBJECTPROP *object prop newvalue*) [NoSpread Function]

As in PLOTPROP. The distinguished props are OBJECTMENU, OBJECTLABEL, and OBJECTDATA. The property, OBJECTMENU, may be set as well as read; if the newvalue is a list of items, it will be coerced into a menu.

(PLOTBJECTADDPROP *object prop itemtoadd firstflg*) [Function]

As in PLOTADDPROP. Firstflg indicates that the new item is to be the first in the list rather than the last.

(PLOTBJECTDELPROP *object prop itemtodelete*) [Function]

As in PLOTDELPROP.

#### DEFAULT MOUSE BUTTON ACTIONS

The user may interact with a plot through its associated window. A plot provides two default menu's, the RIGHT menu, which pops up if the right mouse button is depressed within a plot's window, and typically contains items relevant to the plot as a whole, and the MIDDLE menu, which pops up if the middle mouse button is depressed, and typically contains items relevant to the currently selected plot object. The left mouse button is used exclusively for selection. The right menu may optionally be fixed to the right hand side of the plot window for easy reference. In summary:

##### Left Button

While depressed will select the closest plot object.

##### Middle Button

Pops up a menu of default actions on the selected object

##### Right Button

Pops up a menu of default actions on the plot as a whole

#### DEFAULT MIDDLE MENU ITEMS

##### Label

Label the selected object. Either a default location for the label is selected (for point plot objects), or the user is queried for a location.

##### Unlabel

If the object is label, remove the label.

##### Relabel

Change the object's label

##### Delete

Remove the object from the plot. May be undeleted later.

#### DEFAULT RIGHT MENU ITEMS

##### Layout

Create a SKETCH of the contents of the PLOT. Requires SKETCH and SKETCHSTREAM to be loaded.

##### Redraw

Redraw the plot

**Rescale**

Compute a new scale for both the X and the Y axis based on the objects currently displayed. May also rescale the X or Y axis separately.

**Extend**

Extend the axes slightly on either side so plot objects occurring on the borders may become visible. May be applied separately to either axis.

**Labels**

Change the marginal labels. May either Choose a margin explicitly, or respond to query.

**Tics**

Enable or disable marginal tics.

**Undelete**

Restore the last plot object deleted. Subsidiary items allow selected objects to be restored.

**Deselect**

Deselects the current selected object.

The default menus may be altered or superceded altogether. Each plot object may either use the default middle menu, another cached menu, or provide its own individual menu.

Menus are described by item lists of the form (label function helpstring [(subitems ....)]). Function may be a litatom in which case the function is called with one argument, plot, for right menu items, or two arguments, plotobject and plot, for all other menus. If function is a list the CAR of the list is a APPLIED to (CONS PLOT OBJECT (CONS PLOT (CDR list))), etc.

The following functions facilitate modifying existing menus, and creating new menus.

(PLOTMENU *plot menuname newmenu*) [NoSpread Function]

Plot and menuname are required. If newmenu is not present, then the current value of menu menuname is returned. Menuname may be RIGHT or MIDDLE, in which case the default menus are referred to, or any LITATOM, in which case the cached menu by that name is referred to. Menus other than RIGHT or MIDDLE will typically be specialized menus for particular plot objects. If present, newmenu must be a MENU.

(PLOTMENUITEMS *plot menuname menuitems*) [NoSpread Function]

Plot and menuname are required. If menuitems is not present, then the current item list for the MENU menuname is returned. If menuitems is present, then menu menuname is replaced with a new menu with items list menuitems. All the properties (if any) of the old menu are copied over. Menuname may be one of RIGHT or MIDDLE, in which case the operations refer to the default right or middle mouse button menus or any other LITATOM, in which case the operations refer to a menu cached on plot by that name. Menus other than RIGHT or MIDDLE will typically be specialized menus for particular plot objects.

(PLOTADDMENUITEMS *plot menuname itemstoadd*) [Function]

Itemstoadd must be a list of menu items. Adds each item in itemstoadd to the end of the item list for menu menuname and replaces menu menuname with a new MENU having the appropriate item list.

Returns the the new item list for menuname.

(PLOTDELMENUITEM *plot menuname itemstodelete*) [Function]

Itemstodelete must be a list of items. For each element of itemstodelete, if it is a LITATOM, then deletes the item whose CAR is EQ to it. If it is a LISTP, then deletes the item EQUAL to it. Replaces menu menuname with a new MENU having the appropriate item list.

Returns NIL if no items were deleted, else the new item list.

(PLOT.FIXRIGHTMENU *plot fixedflg*) [NoSpread Function]

Fixedflg is optional. If not present that the current state of the right menu of plot is returned; T implies the right menu is fixed. If Fixedflg is supplied the right menu state is correspondingly changed.

The middle button menu for a particular plot object is a property of that plot object, and may be accessed via the function PLOTOBJECTPROP. For example, the expression,

(PLOTOBJECTPROP object 'OBJECTMENU)

will return the current middle button menu for object. If the OBJECTMENU property is NIL, then the system default MIDDLE menu is used, if it is a LITATOM, than a specialized cached menu by that name is used, finally, if it is a MENU, then that menu is used.

Two default fonts are provided, a large font for labels and a small font for tic marks. Both may be reset and that aspect of a plot will change accordingly with the next redraw.

LARGEPLOTFONT [Variable]

Default value: (Gacha 12 BRR)

SMALLPLOTFONT [Variable]

Default value: (Gacha 8 MRR)

### Detailed Operation

Most visible aspects of a PLOT may be changed programmatically. The following functions allow the user to specify labels, etc., as well as override the default algorithms for drawing tics, etc.

(PLOTLABEL *plot margin newlabel nodrawflg*) [NoSpread Function]

Plot and position are required. Margin must be one of TOP, BOTTOM, LEFT, OR RIGHT. If newlabel is absent, then the current margin label is returned (may be NIL). If newlabel is present then the margin label is set to newlabel. The display is automatically updated unless nodrawflg is non NIL.

(PLOTTICS *plot margin newvalue nodrawflg*) [NoSpread Function]

Plot and margin are required. Margin must be one of TOP, BOTTOM, LEFT, OR RIGHT. If newvalue is absent, returns the tic status of that margin. NIL implies no tics or labels, T implies both. If newvalue is present, then sets margin's tic status. The display is automatically updated unless nodrawflg is non NIL.

The appearance of the tic marks will also depend on the tic generation method employed. The default is simply to make down tics at "pretty" intervals from the max to the min of each axis in world coordinates. However, non-numeric tic marks, and other behaviors are user specifiable by the function PLOTTICMETHOD.

(PLOTTICMETHOD *plot margin newmethod nodrawflg*) [NoSpread Function]

Plot and margin are required. Margin must be one of TOP, BOTTOM, LEFT, OR RIGHT. If newmethod is absent, returns the current tic method for margin margin. Newmethod may be one of NIL, implying the default tic method, a list of CONS pairs ( value . label ), in which case label (if non-NIL) will be printed at value, or a list of numbers, which is equivalent to ((value . value) ...) or a function which will be called with args, margin plotscale plot, and should return a list as above. Plotscale is a datatype which describes the current scale of the plot.

(DEFAULTTICMETHOD *margin plotscale plot*) [Function]

The result depends on the ticinfo field of plotscale, which should be an instance of the PLOTSCALE datatype. The ticinfo field will be an instance of datatype TICINFO. If its ticinc field is a number (the usual case) then it returns a list of numbers, starting at ticmin and ending at ticmax in increments of ticinc, otherwise returns ticinc (should be a list).

When a plot object is added to a plot, the scale of the plot is adjusted so that the object is visible. This is accomplished by comparing the extent (in world coordinates) of the object with the current scale of the plot. If the scale needs to be enlarged, a new interval is chosen for each axis which is guaranteed to include the object and also be some multiple of a "round" increment -- in other words, a pretty tic interval. The default behavior of this scaling algorithm may be altered in several ways.

The pretty tic interval is determined by the TICFN for each axis. The default uses the function SCALE to find a suitable interval. This may be altered by supplying a TICFN other than the default.

Given a pretty tic interval, the default is to simply use the end points of that interval as the endpoints of the scale for each axis. This may be altered by supplying a SCALEFN other than the default.

In other words the actually displayed interval (for each axis) in world coordinates (what I will call the plot interval) is separated from the pretty tic interval (for each axis). The pretty tic interval is computed first, then the plot interval is computed in the presence of that information. This separation is useful if the user wishes to plot objects in a coordinate system different from the one used to display tic marks.

The current state of each axis of a PLOT is cached in the plot property plotscale, whose value is an instance of datatype PLOTSCALE. A PLOTSCALE has three fields for each axis, one which contains an instance of AXISINTERVAL, describing the actual plot interval for that axis, another which contains an instance of TICINFO, which describes the pretty tic interval for that axis, and a third which is a simply a place to cache a user supplied TICFN and SCALEFN.

(PLOTTICFN *plot axis ticfn nodrawflg*) [NoSpread Function]

Ticfn is optional. If not present the current ticfn for the indicated axis is returned. If supplied, the state of that axis is correspondingly updated. A ticfn is called with args min, max, and plot and should return an instance of TICINFO. If the state of plot is changed, the appropriate axis is rescaled. A value of NIL implies the default ticfn.

(DEFAULTTICFN *min max --- --*) [Function]

The default ticfn for each axis. Uses the function SCALE to find a suitable pretty tic interval.

(PLOTSCALEFN *plot axis scalefn nodrawflg*)

[NoSpread Function]

Scalefn is optional. If not present the current scalefn for that axis of plot is returned. If supplied, the state of that axis is updated. A scalefn is called with four arguments, the min and max extent (in world coordinates) on that axis of the plotobjects currently displayed, the TICINFO for that axis, and the plot; the scalefn should return an AXISINTERVAL which will determine the scale for that axis of plot. A value of NIL implies the default scalefn.

(DEFAULTSCALEFN *min max ticinfo*)

[Function]

The default scalefn for each axis.

Returns an AXISINTERVAL with endpoints identical to the endpoints of ticinfo.

(ADJUSTSCALE? *extent plot*)

[Function]

Determines whether extent will fit into the current viewing area of plot. If so, returns NIL. If not, returns T and updates the plotscale of plot.

(EXTENTOFPLOT *plot*)

[Function]

Computes the current extent of plot by mapping EXTENTOBJECT down the display list. Returns an EXTENT.

To be precise, the scaling algorithm operates as follows; a min and max extent of the data is computed (via EXTENTOFPLOT or entered manually in the case manual rescaling), then CHOSETICS is called, which returns an instance of TICINFO. CHOSETICS either uses a default TICFN, or one supplied by the user. The default TICFN, calls SCALE repeatedly to find an "optimal" tic interval in world coordinates. Once the TICINFO instance has been computed, CHOSESCALE is called with the original min, max and the TICINFO, and returns an instance of AXISINTERVAL, which will determine the actually displayed plot interval. Again, CHOSESCALE either uses a default SCALEFN, or one supplied by the user. The default SCALEFN simply uses the end points of the passed in pretty tic interval as the end points of the AXISINTERVAL which it returns. Finally, the PLOT is redrawn with the new scale -- notice that the plot interval may either be larger or smaller than the pretty tic interval; the margin drawing routines are robust enough to deal with all cases.

For example, suppose the world coordinates are in centigrade and it is desired to produce a pretty tic interval in units of Fahrenheit (this is an easy case since the transformation between scales is linear -- more about that later). The user would then supply a TICFN which would transform the incoming min and max to Fahrenheit, apply the default TICFN on the transformed min and max, obtain a TICINFO in Fahrenheit, transform the fields of that record back to Centigrade, and return that record. Note, it is always assumed that the fields of a returned TICINFO are in the units of the world coordinate system. The rest of the machinery would then go through as before.

A trickier example is one in which it is desired to produce unequipped tic marks. Suppose the data were plotted on a log scale (that is, log was applied BEFORE plotting the data). The default algorithm would produce a pretty tic interval in the log scale. It might be desired instead to produce one pretty in the original scale. The user would then supply a TICFN which would exponentiate the incoming min and max, apply the default TICFN on the transformed min and max, obtain a TICINFO in the original scale, then return a TICINFO in the logscale. Note; since equispaced tic marks in the original scale are not equispaced in the log scale, the TICINC field of the returned TICINFO would be a list of the unequipped tic marks values, rather than a number.

The plot scale of each axis may be manipulated directly through the following functions.

**(PLOTAXISINTERVAL *plot axis newinterval nodrawflg*)** [Function]

Plot and axis are required. Axis must be one of X, or Y. If newinterval is NIL , returns the current AXISINTERVAL for that axis. If newinterval is non-NIL it must be an AXISINTERVAL.

**(PLOTTICINFO *plot axis newticinfo nodrawflg*)** [Function]

Plot and axis are required. Axis must be one of X, or Y. If newticinfo is NIL , returns the current TICINFO for that axis. If newticinfo is non-NIL it must be a TICINFO.

On occasion it is useful to clean out an existing plot instead of creating a new one.

**(PLOT.RESET *plot xscale yscale flushmargins flushprops nodrawflg*)** [Function]

Returns plot to a pristine state. If xscale and yscale are provided, the scale of the plot is set accordingly.

Finer control over the behavior of plot objects is possible through the following functions.

**(TRANSLATEPLOT OBJECT *plotobject dx dy plot nodrawflg*)** [Function]

Moves plotobject dx, dy in world coordinates and updates the associated window accordingly. The update is suppressed if nodrawflg is non NIL.

**(DRAWPLOT OBJECT *plotobject plot*)** [Function]

Draw plotobject in the window associated with plot. As with all the display functions, the window should be opened beforehand. DRAWOBJECT does NOT check that the window is open.

APPLY's the plotobject's DRAWFN.

**(ERASEPLOT OBJECT *plotobject plot*)** [Function]

APPLY's the plotobject's ERASEFN

**(HIGHLIGHTPLOT OBJECT *plotobject plot*)** [Function]

Invoked when a plotobject is selected

**(LOWLIGHTPLOT OBJECT *plotobject plot*)** [Function]

Invoked when a plotobject is deselected

**(EXTENTOFPLOT OBJECT *plotobject plot*)** [Function]

Computes the extent of plotobject in world coordinates.

Returns an EXTENT, which has fields MAXX, MINX, etc.

**(DISTANCETOPLOT OBJECT *plotobject streamposition plot*)** [Function]

Returns the "distance" to plotobject from streamposition in stream coordinates. Value returned may be a FIXP or a FLOATP, but is always a distance in stream coordinates.

**(CLOSESTPLOT OBJECT *plot streamposition*)** [Function]

Returns the "closest" plotobject on plot's display list to streamposition.

**(DESELECTPLOT OBJECT *plot*)** [Function]

Deselects the current selected object of plot

Plot objects also have "afterfns". That is, functions which are optionally invoked after some standard operation. These are stored as plot object properties with distinguished names, and invoked with at least two args, the plotobject and the plot.

**WHENADDED**FN [Property]

The **WHENADDED**FN is called with three arguments, plotobject, plot, and nodrawflg

**WHENDELETED**FN [Property]

The **WHENDELETED**FN is called with four arguments, plotobject, plot, nodrawflg, and nosaveflg.

**WHENDRAWN**FN [Property]

The **WHENDRAWN**FN is called with three arguments, plotobject, viewport and plot.

**WHENERASED**FN [Property]

**WHENHIGHLIGHTED**FN [Property]

**WHENLOWLIGHTED**FN [Property]

**WHENTRANSLATED**FN [Property]

A PLOT has two associated windows, the mainwindow in which the graphics, labels, tics, etc. are displayed and an attached promptwindow. The mainwindow is cached as plot property and may be accessed via the function PLOTPROP. A function is provided for easy access to the prompt window.

(**PLOTPROMPT** *text plot*) [Function]

Text is output in the one character high prompt window of plot.

PLOT's may be drawn in ANY imagestream (but only interacted with in the PLOT's associated window). The following function is the fundamental draw primitive.

(**DRAWPLOT** *plot stream streamviewport streamregion*) [Function]

Stream is any imagestream. Streamviewport is a viewport on that stream that defines the the world to stream transformation. Streamregion is a region in stream coordinates that will contain the entire image (for a window it will be the CLIPPINGREGION). Streamviewport is usually the result of **ADJUSTVIEWPORT**.

For more information about viewport, consult the documentation for the **TWODGRAPHICS** module.

(**ADJUSTVIEWPORT** *viewport streamregion plot*) [Function]

Viewport is a **VIEWPORT** whose parentstream is the imagestream of interest. Streamregion is a region in stream coordinates that will contain the entire image.

Adjusts the Streamsubregion and Worldregion of viewport to reflect the current scale and margin setting of plot.

(**MINSTREAMREGIONSIZE** *stream plot*) [Function]

Returns a **CONS** pair (minwidth . minheight) of the plot in stream coordinates.



A plot has "afterfns" for two major operations, opening and closing the plotwindow. These are stored as plot properties with distinguished names. The values of these properties may be a single function or a list of functions which are called in sequence with the plot as an argument.

WHENOPENEDFN [Property]

WHENCLOSEDFN [Property]

PLOT's may be copied, made into image objects, dumped onto files, sent in the mail, etc.

(COPYPLOT *plot*) [Function]

Returns a copy of plot. The user defined properties require special handling. If there exists a plot prop COPYFN, which may be function or list of functions, the function (or functions) will be invoked with the arguments newplot plot and propname for each user defined property on plot. If the function returns a non-NIL value, it will be used as the value of propname on newplot. In the case of a list of functions, the first non-NIL value (traveling from the head to the tail of the list of functions) will be used as the new prop value. Otherwise the prop will be HCOPYALL'ed.

(COPYPLOTOBJECT *plotobject plot*) [Function]

Returns a copy of plotobject. The protocol for copying objectprops is similar to plot props. The plotobject may have a COPYFN prop which may be a function or list of functions. The function (or functions) will be invoked with the arguments newplotobject plotobject plot propname. The first non-NIL value will be used as the prop value else the property will be HCOPYALL'ed.

(PRINTPLOT *plot stream*) [Function]

Writes out an HREADable symbolic representation of plot on stream. Again, user defined properties require special handling. If there exists a plot prop PUTFN, which may be function or list of functions, the function (or functions) will be invoked with the arguments plot propname and stream for each user defined property on plot. If the function returns a non-NIL value, it is assumed an HREADable representation of the prop value has been written out on stream. In the case of a list of functions, the functions will invoked one at time, starting from the head of the list, until a non-NIL result is obtained. If there is no PUTFN, or the function (or none of the functions) returns a non-NIL value, the prop is HPRINT'ed.

Lists of the form ((FUNCTION function) arg) are recognized by the inverse of PRINTPLOT, READPLOT, to imply that function should be called with plot and arg as arguments at HREAD time, and the value returned to be the prop value.

(PRINTPLOTOBJECT *plotobject plot stream*) [Function]

Writes out an HREADable symbolic representation of plotobject on stream. As in PRINTPLOT user defined object properties require special handling. The protocol is the same as in PRINTPLOT.

The following data types have HPRINT macros and need no special handling: FONTDESCRIPTOR, MENU, PLOT, and PLOTOBJECT.

A file package command has been defined to simplyfy dumping PLOT's on files.

(PLOTS . *plots*) [FilePkgCom]

The syntax is identical to VARS.

A plot image object is fully supported.

(CREATEPLOTIMAGEOBJ *plot*)

[Function]

Returns an image object which contains a copy of plot. These image objects can also be created by copy-selecting from a plot window into a host window (e.g. TEdit or Sketch) that supports image objects. Such a selection will ask whether the plot should be inserted as a bitmap or a plot, the latter case constructing a plot image object. Buttoning on the image object provides the option of reshaping the plot or creating a separate plot window in which the plot can be modified. Closing the plot window will ask whether the new plot should be reinserted in the host.

---

---

**PLOT EXAMPLES**

---

---

By: Jan Pedersen (Pedersen.PA @ Xerox.com)

Uses: PLOT

This module contains two examples of how PLOT might be used to produce high level plotting facilities. The first example is a histogram primitive, and the second is a scatterplotter. The code is commented, and exploits most of the facilities in PLOT. The scatterplot example is the simpler of the two, and is suggested as a starting point.

(SCATPLOT *y x pointlabels ylabel xlabel title symbol*) [Function]

Generates of a scatterplot of *y* vs *x* which are numeric lists of equal length. If *x* is NIL, then *y* is plotted vs the integers from 1 to (LENGTH *y*). *Pointlabels* is a list of labels, one for each point plotted. *Ylabel* and *xlabel* are labels for the *x* and *y* axis respectively. *Title* is a title for the scatterplot. *Symbol* is the plotting symbol to use, must be a BITMAP; defaults to STAR.

Returns a PLOT.

(HISTPLOT *batch label shade*) [Function]

*Batch* is a list of numbers, or a list of pairs (number . frequency) whose histogram will be displayed. *Label* is an optional label for those numbers. *Shade* is a shade to use to fill the bars of the histogram (defaults to SHADE3). The case of all entries in *batch* being integers is treated specially.

Returns a PLOT.

---



---

## PLOT OBJECTS

---



---

By: Jan Pedersen (pedersen.PA @ Xerox.com)

Uses: PLOT and TWODGRAPHICS

Plot objects are the primitive quantities of the PLOT module. A plot object is abstracted as an instance of datatype PLOT OBJECT. A point plot object is an instance of PLOT OBJECT whose data component describes a point. That is, a point plot object is a subtype of PLOT OBJECT; all plot objects satisfy (`type? PLOT OBJECT F00`) but only a point plot object satisfies in addition (`PLOT OBJECT SUBTYPE? POINT F00`).

A PLOT OBJECT is both a datatype and a collection of functions that implements a set of generic operations on that plot object. A plot object must know how to draw itself, erase itself, highlight itself, etc. The PLOT module then deals only with generic operations, and allows the plot objects to implement them as is appropriate.

PLOT OBJECT	[Datatype]
OBJECTFNS	[Field]
Must be an instance of PLOT FNS	
OBJECTSUBTYPE	[Field]
Describes the plot objects subtype	
OBJECTUSERDATA	[Field]
Space for a property list	
OBJECTMENU	[Field]
The object's MENU	
OBJECTLABEL	[Field]
Something to print	
OBJECTDATA	[Field]
Space for a datatype that describes the subtype of this PLOT OBJECT	
The field OBJECTFNS must be an instance of PLOT FNS, essentially a vector of functions which implements the generic operations.	
PLOT FNS	[Datatype]
DRAWFN	[Field]
Implements the DRAW OBJECT generic operation	
ERASEFN	[Field]
etc.	

HIGHLIGHTFN	[Field]
LOWLIGHTFN	[Field]
LABELFN	[Field]
MOVEFN	[Field]
EXTENTFN	[Field]
DISTANCEFN	[Field]
COPYFN	[Field]
PUTFN	[Field]
GETFN	[Field]

The generic operations are:

(DRAWPLOT OBJECT *object viewport plot*) [Function]

Draw the object within viewport. A VIEWPORT may be thought of as a sub imagestream. It will usually be associated with the plot's PLOTWINDOW, but might also be associated with some other image stream. Typically this generic operation will make use of functions from TWODGRAPHICS and the position of the object in world coordinates. The plot is also passed as an argument, so that the draw operation may make use of information cached on the property list of plot.

The only operation that is expected to draw on streams other than the PLOTWINDOW is drawobject, so the drawfn may have to behave differently depending on the imagestreamtype of the stream. All other generic operations are assumed to operate on the PLOTWINDOW. The idea here is that plot's may be drawn on any stream, but may be interacted with only through the PLOTWINDOW. It is also guaranteed that an object will be drawn before it is erased, highlighted, etc.

(ERASEPLOT OBJECT *object viewport plot*) [Function]

Erase the object from the viewport. The inverse of DRAWOBJECT. It is guaranteed that the viewport will be on the PLOTWINDOW

(HIGHLIGHTPLOT OBJECT *object plot*) [Function]

Highlight the object. Used in selection.

(LOWLIGHTPLOT OBJECT *object plot*) [Function]

The inverse of HIGHLIGHTOBJECT. With XOR drawing the HIGHLIGHTFN and the LOWLIGHTFN can often be the same.

(MOVEPLOT OBJECT *object dx dy plot*) [Function]

Destructively alter the object's OBJECTDATA, so that its position is moved dx, dy units (in world coordinates).

(LABELPLOT OBJECT *object plot*) [Function]

If it is desired to label the object, the LABELFN will be called. Often the function LABELGENERIC will do the trick.

(EXTENTOFPLOT OBJECT *object plot*) [Function]

Should return an EXTENT, which expresses the range of the object in world coordinates.

EXTENT [Datatype]

MINX [Field]

Minimum extent in the X (horizontal) direction

MAXX [Field]

Maximum extent in the X (horizontal) direction

MINY [Field]

Minimum extent in the Y (vertical) direction

MAXY [Field]

Maximum extent in the Y (vertical) direction

All fields are type floating.

(DISTANCETOPLOT OBJECT *object streamposition plot*) [Function]

Should return a number (more efficient if it returns a SMALLP), which is some measure of the distance from the REPRESENTATION of the object to the POSITION streamposition. Note that distance is calculated in stream coordinates, NOT world coordinates. This is done for efficiency and logical consistency. Selection makes most sense as an activity in stream coordinates.

A plot object will typically cache its stream coordinates when it is drawn. Although not strictly necessary (it is always possible to backsolve to stream coordinates from world coordinates), this improves efficiency many fold by avoiding generation of floating point boxes.

The following functions are provided to allow the plot object to customize how it is copied, printed on file, etc. The generic defaults will usually be satisfactory.

(COPYPLOT OBJECT *object plot*) [Function]

Returns a copy of object. COPYOBJECT will create a new instance of PLOT OBJECT and copy over all the fields of object except for OBJECTDATA. The object's COPYFN is evoked with the arguments object and plot and is expected to return a new instance of OBJECTDATUM. The object's property list is handled as follows: If object has a prop COPYFN (which may be a function or list of functions), for each property it is called with the arguments newobject, oldobject, plot, proptype. If the returned value is non-nil it is used as the value for that property on newobject; else the prop value is H COPYALL'ed. If the value of COPYFN is a list of functions, they are invoked in order head to tail, and the first non-NIL value is used as the new value.

(PRINTPLOT OBJECT *object plot stream*) [Function]

Writes out to stream an HREADable symbolic representation of object. As in COPYOBJECT, PRINTOBJECT takes care of all PLOT OBJECT fields except of OBJECTDATUM. The object's PUTFN will be invoked with the arguments object plot stream and is expected to write out a representation of OBJECTDATUM which is HREADable. This will usually be in prop list format.

Again the prop list of object requires special handling. The special object prop PUTFN may be a function or list of functions. For each property it will be invoked with the arguments object plot proptype and stream and if it returns a non-NIL value, it is assumed that property has been

written out in a HREADable format. Again, if the the PUTFN prop is a list of fns then if any one of them returns non-NIL then the property is assumed written out. If there is no PUTFN then the property is (HPRINT prop stream NIL T) 'ed.

PUTFNS may put out special lists of the form ((FUNCTION ffname) arg) in which case ffname will be invoked at HREAD time with args object plot propname arg and ffname will be expected to return the propvalue of propname.

(READPLOTOBJECT *stream*) [Function]

Reads in the product of PRINTOBJECT. Calls the objects GETFN to read in the OBJECTDATA field.

An instance of PLOTFNS may be created by the function:

(CREATEPLOTFNS *drawfn erasefn extentfn distancefn highlightfn  
lowlightfn labelfn movefn copyfn putfn getfn borrowfrom*) [Function]

Returns an instance of PLOTFNS. Drawfn, erasefn, and extentfn are required. If a distancefn is supplied then so must be a highlightfn. Lowlightfn defaults to highlightfn, labelfn defaults to LABELGENERIC. The other arguments also default to some safe, if not too efficient genericfn.

A primitive inheritance scheme is implemented via the optional argument borrowfrom. If supplied, borrowfrom must be an instance of PLOTFNS. Before creating the new instance of PLOTFNS, the NIL arguments passed are filled in from the fields of borrowfrom, with the following exception; lowlightfn is only inherited if highlightfn is also NIL.

The OBJECTDATA field will typically be a datatype which holds the data characterizing the PLOTOBJECT. For example a point plot object will have an OBJECTDATA field whose value is an instance of the datatype POINTDATA (has fields position, symbol, etc). So, a point PLOTOBJECT is a specialization of PLOTOBJECT. The field OBJECTSUBTYPE is supplied to make the subtype explicit. The following macro is provided to facilitate testing for plot object subtypes.

(PLOTOBJECTSUBTYPE? *subtype plotobject*) [Macro]

Essentially tests if (EQ subtype (fetch OBJECTSUBTYPE of plotobject))

(PLOTOBJECTSUBTYPE *plotobject*) [Function]

Returns the value of the OBJECTSUBTYPE field.

PLOTOBJECTS may be created via the function:

(CREATEPLOTOBJECT *objectfns objectlabel objectmenu objectdata*) [Function]

Returns an instance of PLOTOBJECT. Coerces objectmenu into a MENU if it is an item list.

The following subtypes of PLOTOBJECT are currently implemented.

pointPLOTOBJECT, curvePLOTOBJECT, polygonPLOTOBJECT, linePLOTOBJECT,  
graphPLOTOBJECT, texttPLOTOBJECT, filledrectanglePLOTOBJECT, compound PLOTOBJECT

The functions CREATEPOINT, etc. return an instance of PLOTOBJECT, with the appropriate OBJECTFNS and OBJECTDATA. In order for this to work, some intializations must be done at load time.

The function PLOT.SETUP performs the intializations at LOAD time.

(PLOT.SETUP *opstable*)

[Function]

Opstable must be a list of lists of the form:

```
(  
(subtype1 (opname1 function1) (opname2 function2) ....  
(subtype2 (opname1 function1) (opname2 function2) ....  
.....  
(subtypenamen (opname1 function1)(opname2 function2) ....  
)
```

Creates one instance of PLOTFNS for each subtype name.

In summary, to add a new plot object you need to:

- Determine the data required to describe the new subtype. This may involve declaring a new datatype.
- Write functions similar to CREATEPOINT and PLOTPOINT for the new subtype.
- Write (or borrow) the functions which implement the generic ops described above.
- Invoke MAKEPLOTFNS to create an instance of PLOTFNS for the new plot object subtype, which all objects of that subtype will refer to.
- If continued use of the new plot object is contemplated, PLOT.SETUP should be evoked at load time to effect the proper initializations.

Look at the code for existing plot objects for more details. The point plot object is the simplest example.



---



---

## PLOTOBJECTS1

---



---

By: Tad Hogg (hogg.PA @ Xerox.com)

Uses: PLOT and PLOTOBJECTS

PLOTOBJECTS1 defines additional plot objects for use with PLOT.

### NEW PLOTOBJECTS

ERRORPOINT - a point with vertical and/or horizontal error bars.

SAMPLESET - a set of points drawn as line segments to a specified vertical or horizontal line.

### FUNCTIONS

The following functions provide an add facility for the new objects. They are similar to the corresponding functions for the standard plot objects, e.g. PLOTPOINT, etc. The allowed forms of the arguments *symbol*, *style*, *menu* and *nodrawflg* are the same as for the standard functions.

(PLOTERRORPOINT *plot position-range label symbol style menu nodrawflg*) [Function]

Position-range is a list of the form (POSITION XRANGE YRANGE). POSITION is the position of the point in world coordinates. XRANGE and YRANGE control the length of the horizontal and vertical error bars respectively. If the range is NIL, no error bars are drawn. If it is a number, it is a distance (in world coords) for the error bar to extend on each side of the point. Finally, if it is a pair of numbers (NegDist . PosDist) it specifies the extent of the error bar in the negative and positive directions, respectively. Symbol is used to plot the point. Symbol defaults to STAR. Style specifies the style to use for drawing the error bars.

Returns an ERRORPOINT PLOT OBJECT.

(PLOTERRORPOINTS *plot position-ranges labels symbol style menu nodrawflg*) [Function]

As above except that position-ranges is a list of POSITION-RANGES as described above and labels may also be a list. Reasonable things happen if positions and labels are of unequal length.

Returns a list of ERRORPOINT PLOT OBJECT's.

(PLOTSAMPLESET *plot positions constant vertical? side label style menu nodrawflg*) [Function]

The list of POSITION's defines a number of sample points. Constant specifies the location of a vertical or horizontal line, depending on whether vertical? is non-NIL. Line segments are drawn from the sample points to this line. Side determines which points are actually included. If side is NIL, only those points whose coord is greater than constant will be drawn (i.e. points above or to the right of the line). If side is T, only those with coord less than constant will be drawn. Otherwise, all points will be included. Style specifies the style to use for drawing the line segments.

Returns a SAMPLESET PLOT OBJECT.

All plot objects may be created independently of the previous functions. This is useful if it is desired to create a plot object without entering it on a PLOT's display list. The following functions create and return the new plot objects.

(CREATEERRORPOINT *position-range label symbol style menu*) [Function]

Returns an ERRORPOINT PLOT OBJECT.

(CREATESAMPLESET *positions constant vertical? side label style menu*) [Function]

Returns a SAMPLESET PLOT OBJECT.

In addition there are a number of functions to aid in creating position-ranges used with the error point plot objects:

(MAKE-POSITION-RANGE *position xrange yrange*) [Function]

Returns a position-range suitable for use for specifying an error point. The arguments are as described above for PLOTERRORPOINT.

(LOG-ERROR-RANGE *position-range axis base*) [Function]

Returns a position-range corresponding to *position-range* converted to a log scale. *base* is the log base to use (defaults to 10) and *axis* specifies which axis to convert: :X or :Y for a specific axis, NIL for both. Note that the position, with its error bars must be positive in order to be converted to a log scale.

(LOG-ERROR-RANGE-LIST *position-ranges axis base*) [Function]

Converts a list of *position-ranges* to log scale.

---



---

## POSTSCRIPT

---



---

By: Matt Heffron (BEC.HEFFRON@ECLA.USC.EDU)

### INTRODUCTION

The PostScript package defines a set of imageops for printers which understand the PostScript page description language by Adobe. At Beckman we have successfully used TEdit, Sketch, LISTFILES, and HARDCOPYW to an Apple LaserWriter and an AST TurboLaser PS. The PostScript imagestream driver installs itself when it is loaded. All symbols in the PostScript driver are located in the INTERLISP: package.

### VARIABLES

POSTSCRIPT.FONT.ALIST [InitVariable]

POSTSCRIPT.FONT.ALIST is an ALIST mapping Lisp font names into the root names of PostScript font files. It is also used for font family coercions. The default value should be acceptable for any of the fonts which are built into the Apple Laserwriter.

POSTSCRIPTFONTDIRECTORIES [InitVariable]

POSTSCRIPTFONTDIRECTORIES is the list of directories where the PostScript .PSCFONT font files can be found. The default value is: (" {DSK}<LISPFILES> FONTS>PSC> ").

\POSTSCRIPT.SHORTEDGE.SHIFT [InitVariable]

\POSTSCRIPT.SHORTEDGE.SHIFT is the distance (in points) to shift the image perpendicular to the short edge of the paper. A positive value gives a shift upward in portrait mode, and to the right in landscape mode. The default value is: 0.

\POSTSCRIPT.LONGEDGE.SHIFT [InitVariable]

\POSTSCRIPT.LONGEDGE.SHIFT is the corresponding variable for shifts perpendicular to the long edge of the paper. A positive value here gives a shift to the right in portrait mode and downward in landscape mode. The default value is: 0.

\POSTSCRIPT.SHORTEDGE.PTS [InitVariable]

\POSTSCRIPT.SHORTEDGE.PTS indicates the printable region of the page (in points) along the short edge of the paper. It should be adjusted to allow for any shifts of the image (see above). The default value is: 576 (= 8 inches).

\POSTSCRIPT.LONGEDGE.PTS [InitVariable]

\POSTSCRIPT.LONGEDGE.PTS indicates the printable region of the page (in points) along the long edge of the paper. It should be adjusted to allow for any shifts of the image (see above). The default value is: 786.24 (= 10.92 inches).

### HINT

The AST TurboLaser PS has an imageable area on the page which is a different size than that of the Apple LaserWriter. The values of

`\POSTSCRIPT.SHORTEGE.PTS` and `\POSTSCRIPT.LONGEDGE.PTS` for the AST are 575.76 and 767.76, respectively.

`\POSTSCRIPT.MAX.WILD.FONTSIZE` [InitVariable]

`\POSTSCRIPT.MAX.WILD.FONTSIZE` indicates the maximum point size that should be returned from `FONTSAVAILABLE` when the `SIZE` argument is wild (i.e. \*). All integer point sizes from 1 to `\POSTSCRIPT.MAX.WILD.FONTSIZE` will be indicated as available. The default value is: 72.

`POSTSCRIPT.PREFER.LANDSCAPE` [InitVariable]

`POSTSCRIPT.PREFER.LANDSCAPE` indicates if the `OPENIMAGESTREAM` method should default the orientation of output files to `LANDSCAPE`. The default value is: `NIL`.

`POSTSCRIPT.TEXTFILE.LANDSCAPE` [InitVariable]

`POSTSCRIPT.TEXTFILE.LANDSCAPE` indicates if the printing of `TEXT` files (e.g. `LISTFILES`, ...) should force the orientation of output files to `LANDSCAPE`. The default value is: `NIL`.

`POSTSCRIPT.BITMAP.SCALE` [InitVariable]

`POSTSCRIPT.BITMAP.SCALE` specifies an independent scale factor for display of bitmap images (e.g. window hardcopies). Values less than 1 will reduce the image size. (i.e. a value of 0.5 will give a half size bitmap image.) The position of the scaled bitmap will still have the `SAME` lower-left corner (i.e. the scaled bitmap is not centered in the region of the full size bitmap image). The default value is: 1.

#### HINT

Setting `POSTSCRIPT.BITMAP.SCALE` to 0.96, instead of 1, will give cleaner `BITMAP` images on a 300 dpi printer. (This corrects for the 72 ppi imagestream vs. the 75 dpi printer, using 4x4 device dots per bitmap pixel.) Also, values of 0.24, 0.48 and 0.72, instead of 0.25, 0.5 and 0.75, will also give cleaner images for reduced size output. In general, use integer multiples of 0.24 for a 300 dpi printer.

`POSTSCRIPT.TEXTURE.SCALE` [InitVariable]

`POSTSCRIPT.TEXTURE.SCALE` specifies an independent scale for the display of bitmap textures. The value represents the number of device space units per texture unit (bitmap bit). The default value is 4, which represents each bit of the texture as a 4x4 block, so that textures are approximately the same resolution as on the screen (for 300 dpi output devices, such as the Apple Laserwriter).

The PostScript package extends the allowed representations of a texture, beyond 16-bit `FIXP` and 16x16 bitmap, to ANY square bitmap. (If the bitmap is not square, its longer edge is truncated from the top or right to make it square.) Use this feature with caution, as large bitmap textures, or sizes other than multiples of 16 bits square, require large amounts of storage in the PostScript interpreter (in the printer controller), and can cause `limitcheck` errors when actually printing.

Anywhere that a texture or color can be used on an imagestream or in the specification of a `BRUSH`, you can instead give a `FLOATP` between 0.0 and 1.0 (inclusive) to represent a PostScript halftone gray shade. (0.0 is black and 1.0 is white. Specifically, the value sets the brightness of the shade.) The value you specify will not be range checked, and will be passed directly through to the PostScript `setgray` operator. (E.g. you can pass 0.33 as the color to `DRAWLINE` to get a dark gray line with approximately 67% of the pixels in the line black.)

`POSTSCRIPT.IMAGESIZEFACTOR` [InitVariable]

`POSTSCRIPT.IMAGESIZEFACTOR` specifies an independent factor to change the overall size of the printed image. This re-sizing affects the entire printed output (specifically, it superimposes its effects upon those of `POSTSCRIPT.BITMAP.SCALE` and `POSTSCRIPT.TEXTURE.SCALE`). Values

greater than 1 enlarge the printed image, and values less than 1 reduce it. An invalid POSTSCRIPT.IMAGESIZEFACTOR (i.e. not a positive, non-zero number) will use a value of 1. The BITMAPSCALE function for the POSTSCRIPT printer type does NOT consider the POSTSCRIPT.IMAGESIZEFACTOR when determining the scale factor for a bitmap.

### MISCELLANEOUS

The SCALE of a PostScript imagestream is 100. This is to allow enough resolution in the width information for fonts to enable TEdit to correctly fill and justify text.

The first time any PostScript imagestream is created (even if only to hardcopy a bitmap or window) the DEFAULTFONT is instantiated (unless a FONTS option was given to the OPENIMAGESTREAM, in which case the initial font for the imagestream will be set to that font, or to the CAR if a list).

The PostScript imagestream method for FILLPOLYGON uses the global variable FILL.WRULE as the default value for the WINDINGNUMBER argument. (This is the same variable which is used by the DISPLAY imagestream method for FILLPOLYGON.)

The PostScript imagestream method for OPENIMAGESTREAM (and, therefore, SEND.FILE.TO.PRINTER), supports an IMAGESIZEFACTOR option to change the size of the printed image. The IMAGESIZEFACTOR re-sizing is combined with the POSTSCRIPT.IMAGESIZEFACTOR to produce an overall re-sizing of the printed image. A HEADING option is also supported to give a running header on each page of output. The value of the HEADING option is printed at the top left of the page, followed by "Page " and the appropriate page number. They are printed in the DEFAULTFONT (unless a FONTS option was given to the OPENIMAGESTREAM, in which case it will be that font, or to the CAR if a list).

The PostScript package is contained in the files: POSTSCRIPT.LCOM & PS-SEND.LCOM, with the source in the files: POSTSCRIPT & PS-SEND. The module PS-SEND.LCOM is required and will be loaded automatically when POSTSCRIPT.LCOM is loaded. It contains the function which is called by SEND.FILE.TO.PRINTER to actually transmit the file to the printer. It is, by its nature, quite site specific, so it is in a separate file to make modifying it for any site relatively simple. System record declarations required to compile POSTSCRIPT can be found in EXPORTS.ALL.

I'm pretty sure that the output generated by the PostScript imageops fully conforms to the Adobe Systems Document Structuring Conventions, Version 2.0, January 31, 1987.

### Including Other PostScript Operations

If you wish to insert your own specific PostScript operations into a PostScript imagestream, you can do so with the following functions:

(POSTSCRIPT.OUTSTR *STREAM STRING*) [Function]

POSTSCRIPT.OUTSTR outputs a string or value to the imagestream. *STREAM* must be an open PostScript imagestream. *STRING* is the value to output (STRINGP and LITATOM are most efficient, but any value can be output (its PRIN1 pname is used)).

(POSTSCRIPT.PUTCOMMAND *STREAM STRING<sub>1</sub> ... STRING<sub>n</sub>*) [NoSpread Function]

POSTSCRIPT.PUTCOMMAND is more general for sequences of commands and values. It calls POSTSCRIPT.OUTSTR repeatedly to output each of the *STRING<sub>i</sub>* arguments to *STREAM*.

(\POSTSCRIPT.OUTCHARFN *STREAM CHAR*) [Function]

\POSTSCRIPT.OUTCHARFN is used to output the characters forming the text of a PostScript string (e.g. the argument to a show or charpath operator). *STREAM* is the open PostScript imagestream to output to, and *CHAR* is the CHARCODE of the character to output. The / (slash), ( and ) (parenthesis) characters will be quoted with /, and characters with ASCII values less than 32 (space) or greater than 126 (tilde) will be output as /nnn (in octal). \POSTSCRIPT.OUTCHARFN will output

the ( character to open the string, if necessary. Use POSTSCRIPT.CLOSESTRING (below) to close the string.

(POSTSCRIPT.CLOSESTRING *STREAM*)

[Function]

POSTSCRIPT.CLOSESTRING closes a PostScript string (e.g. the argument to a show or charpath operator). *STREAM* is the open PostScript imagestream. It is important to use POSTSCRIPT.CLOSESTRING to output the ) character to close the string, because it also clears the stream state flag that indicates that a string is in progress (otherwise, the next POSTSCRIPT.PUTCOMMAND would output the commands to close the string and show it).

### Warning

Do not attempt to create a PostScript font larger than about 600 points, as much of Interlisp's font information is stored in SMALLP integers, and too large a font would overflow the font's height, or the width for any of the wider characters. (I know that 600 points is a ridiculously large limit (about 8.3 inches), but I thought I'd better mention it, or someone might try it!)

### Changes from the Lyric Release

The Medley release of this PostScript imagestream driver changed the default value of POSTSCRIPT.TEXTFILE.LANDSCAPE from T to NIL. It also added the support for the HEADING option.

### Known Problems/Limitations

The output generated for a PostScript imagestream is rather brute force. It isn't particularly careful to generate the smallest output file for a given sequence of operations. Specifically, it often generates extra end-of-lines between PostScript operator sequences (this has no effect on the printed output, only on the file size).

Using BITMAPs or Functions as BRUSH arguments to the curve drawing functions is not supported, nor is using a non-ROUND BRUSH with DRAWCIRCLE or DRAWELLIPSE.

There is no support for NS character sets other than 0, and there is no translation of the character code values from NS encoding to PostScript encoding.

There is no support for color.

\POSTSCRIPT.OUTCHARFN is pretty wimpy in its handling of TAB characters. It just outputs 8 SPACES for the TAB.

I haven't yet documented how to build the .PSCFONT files for any new fonts that become available, I'll do that eventually.

---

---

**PS-SEND**

---

---

By: Matt Heffron (BEC.HEFFRON@ECLA.USC.EDU)

Requires: POSTSCRIPT

The module PS-SEND.LCOM is required by the PostScript ImageStream driver, and will be loaded automatically when POSTSCRIPT.LCOM is loaded. It contains the function (POSTSCRIPT.SEND) which is called by SEND.FILE.TO.PRINTER to actually transmit the file to the printer. It is, by its nature, quite site specific, so it is in a separate file to make modifying it for any site relatively simple.

POSTSCRIPT.SEND can handle the simple cases of copying a file to a spool directory or directly to a specific device (using COPYBYTES). The information about how to send a file to a specific host is expected to be on the SPOOLDIRECTORY, SPOOLFILE, SPOOLOPTIONS, HOST.CONTROL.STRING and HOST.CONTROL.AFTER.STRING properties of the host name. It checks first for the SPOOLFILE property on the host name, which must be a full filename that can be opened (by OPENSTREAM). If there is no SPOOLFILE property, then it checks for a SPOOLDIRECTORY property, if there, it will be concatenated together with a generated filename (by (GENSYM USERNAME)) and a ".PS" extension. If either the SPOOLFILE or SPOOLDIRECTORY properties exist, then an output stream will be opened onto the specified file. The value of the SPOOLOPTIONS property on the host name (if any) will be passed as the PARAMETERS argument to OPENSTREAM, and must be an appropriately formed list. (This is useful for cases where the specified destination is an 11xx device, such as {TTY} or {RS232} where you must set additional attributes of the stream like baud rate, etc.) If there is no SPOOLFILE or SPOOLDIRECTORY properties, then nothing will be sent and a message will appear in the PROMPTWINDOW ("Unable to send FILE to HOST.>").

After the output stream is opened, if there is a HOST.CONTROL.STRING property of the hostname, then that string will be printed (IL:PRIN1) to the output stream first, then the first line of the file being sent (for a file generated by the PostScript ImageStream driver, this is the "%! ..." line), then the value of the POSTSCRIPT.CONTROL.STRING from the PRINTOPTIONS argument to POSTSCRIPT.SEND, finally the rest of the input file. (The idea of the HOST.CONTROL.STRING is that it should be a string to control the printing host itself, or perhaps a routing device that is mid-stream between the 11xx and the printer itself. For example, using a SPOOLFILE of "{TTY}FOO.PS" and having the PostScript printer shared by several additional computers (e.g. PC's) by use of a device like the Logical Connection from Fifth Generation Systems, it might be necessary to send a command to the Logical Connection to specify to route the output from this input to the output which is the PostScript printer.) Likewise, if there is a HOST.CONTROL.AFTER.STRING property of the hostname, then that string will be printed to the output stream last, just before closing the stream.

---

---

**PS-TTY**

---

---

By: Matt Heffron (BEC.HEFFRON@ECLA.USC.EDU)

Requires: POSTSCRIPT, PS-SEND, DLTTY

The module PS-TTY defines a printing host named PS-TTY which sends PostScript output to a printer over the {TTY} port of the 11xx. It also puts a function onto AROUNDXITFNS which reinitializes the {TTY} after returning from LOGOUT. The BaudRate and other parameters of the {TTY} port are controlled by the following variables.

**VARIABLES**

PS-TTY-BAUD [InitVariable]

This is the BaudRate for the {TTY} port output stream. Defaults to: 4800.

PS-TTY-DATABITS [InitVariable]

This is the BitsPerSerialChar for the {TTY} port output stream. Defaults to: 8.

PS-TTY-PARITY [InitVariable]

This is the Parity for the {TTY} port output stream. Defaults to: NONE.

PS-TTY-STOPBITS [InitVariable]

This is the NoOfStopBits for the {TTY} port output stream. Defaults to: 1.

PS-TTY-FLOWCONTROL [InitVariable]

This is the FlowControl for the {TTY} port output stream. Defaults to: XOnXOff.



---

---

**PREEMPTIVE**

---

---

By: Larry Masinter (Masinter.pa@Xerox.com)

This module turns on pre-emptive process scheduling. Using IL:\PERIODIC.INTERRUPT, it forces a block in whatever process is running.

(IL:PREEMPTIVE &OPTIONAL STATE)

[Function]

The function PREEMPTIVE turns preemptive process scheduling on and off. (IL:PREEMPTIVE ':ON) turns it on, (IL:PREEMPTIVE ':OFF) turns it off. (IL:PREEMPTIVE) with no argument returns the current state with no change.

**WARNING WARNING WARNING WARNING DANGER DANGER DANGER DANGER**

PREEMPTIVE is dangerous. Many places in the system do not have monitor locks and other mechanisms to prevent one process from overwriting the data of another in the face of preemptive interrupts. (Most do, of course.)

I've run with preemptive scheduling turned on for weeks, and about once a day, my screen gets trashed, windows and menus overwritten, etc. This version of PREEMPTIVE is a little more conservative than previous versions, e.g., it checks to see if the system is running in the MENU code and doesn't do a process switch. However:

**USE AT YOUR OWN RISK. CAUTION CAUTION.**

**NOTE: Using SPY turns preemptive scheduling OFF.**

---



---

## PRESSFROMNS

---



---

By: Tad Hogg (Hogg.pa@Xerox.com)

### INTRODUCTION

This module is a patch to allow Press printers to print NS characters by translating them to appropriate Press fonts. Before loading this file, make sure there are no open press streams (i.e. no hardcopy in progress to a Press printer).

### CONTROLLING CHARACTER SET TRANSLATIONS

The translations are controlled by a number of variables and functions described below. These variables can be modified to provide additional or different translations.

#### Global character set translations

##### NSTOASCIITRANSLATIONS

[Variable]

an ASSOC list whose elements have the form (charset translationArrayName). This specifies which translation array is to be used when translating the specified character set.

Example: ((0 ASCIIFROM0ARRAY) (38 ASCIIFROM38ARRAY)) specifies that ASCIIFROM38ARRAY is bound to the array to be used for translating charset 38.

The translationArrayName is bound to an array whose index ranges from 0 to 255. Each element of the array specifies the translation to use for the corresponding charcode in this charset.

Translations are of one of the following forms:

1. NIL -- no translation specified which will use the font as is for charset 0 and otherwise print a black box to indicate the NS character could not be translated
2. an integer in the range 0 to 255 which indicates that this value is to be used as the translated charcode, but that no font translation is required [This is mainly useful for converting NS to ASCII in charset 0.]
3. a two element translation list of the form (fontFamily charcode) which indicates that this character should be translated to charcode in a font whose family is fontFamily. Note that charcode should be in the range 0 to 255. fontFamily can also be a font descriptor or a font specification list (e.g. (Gacha 10)) of a form acceptable to FONTCREATE.

##### PRESSFONTFAMILIES

[Variable]

a list whose elements are of the form (FAMILY . specialTranslations). The optional list specialTranslations specifies translations to use for Press fonts in charset 0. Each element is (charcode translation) which specifies that translation is to be used for charcode in this family.

Example: ((GACHA (50 (HELVETICA 40)) (51 (TIMESROMAN 45))) (TIMESROMAN) (SYMBOL))

Note: these translations are cached in the font descriptor when PRESS fonts are created so any changes to these variables will not change the translations in previously created fonts.

### Translations in individual fonts

The following two functions provided detailed control over the translations used in individual fonts.

(GETCHARPRESSTRANSLATION *CHARCODE FONT*) [Function]

returns the translation (a two element list) used for CHARCODE in FONT

(PUTCHARPRESSTRANSLATION *CHARCODE FONT NEWTRANSLATION*) [Function]

sets the translation to be used for CHARCODE in FONT. NEWTRANSLATION should be a translation in one of the forms described above.

### Additional functions

(PRESS.NSARRAY *CHARSETFAMILY ASCIARRAY*) [Function]

This function returns a suitable translation array built as the inverse of a translation array from Press to NS characters. Such arrays are used to print Press fonts on Interpress printers and are listed in the variable ASCIIOTSTRANSLATIONS. CHARSET is the character set for which to create a translation. FAMILY is the press font family for which ASCIARRAY is the translation to NS characters. If ASCIARRAY is not specified, the function looks through all arrays included on ASCIIOTSTRANSLATIONS to fill in the translation array. The following two functions provided detailed control over the translations used in individual fonts.

### CONTROLLING PRESS FONT COERCIONS

There is also a mechanism for determining which press font is actually used for the translation. For example, an NS character in the Modern 8 font might translate to an ASCII character in Symbol 8. If this font does not exist on the printer, a (generally incorrect) font change will be done by the printer. The following procedure changes the actual font used, e.g. to Symbol 10 in this example.

The coercion is controlled by a coercion list which is an alist indexed by device. The font coercion scans down the element on the list for the requested device (e.g. PRESS) looking for the first entry that matches the user request. If a match is found, then the entry tells how to construct an appropriate new name from the requested specification. Fields of the newname not specified in the entry are simply copied over.

FONTCOERCIONS [Variable]

This list allows the user to coerce fonts that he knows don't exist on the printer even tho the fonts-widths files doesn't indicate that (e.g. the desired size doesn't exist). FONTCOERCIONS is initialized simply to take all SYMBOL fonts of size less than 10 into 10, and size greater than 12 into 12.

MISSINGFONTCOERCIONS [Variable]

If the initial coerced (or uncoerced) lookup fails, then MISSINGFONTCOERCIONS is used. This takes MODERN into HELVETICA etc--the standard press coercions.

The procedure for determining whether a user request matches a coercion entry is straightforward. If the match-part of the coercion entry is an atomic family name, it matches if it is eq to the requested family. Otherwise, the match-part must be a list of family, size, face in

standard fontname order. If a component is NIL or missing, then it is assumed to match. The only funniness is in size matching. The size component can be NIL (matches anything), a particular size (EQ matches), or a list of the form (< n) or (> n) where n is a size number. The first matches any requested size less than n, and the second matches any requested size greater than n.

FONTCOERCIONS for PRESS starts out as

```
((SYMBOL (< 10) ) (SYMBOL 10)) (SYMBOL (> 12))(SYMBOL 12))
```

MISSINGFONTCOERCIONS for PRESS starts out as

```
((MODERN HELVETICA)(CLASSIC TIMESROMAN) etc.)
```

---

---

**PRETTYFILEINDEX**

---

---

By: Bill van Melle (vanMelle.PA@Xerox.com)

**INTRODUCTION**

PRETTYFILEINDEX is a program for generating indexed listings for Lisp source files. PRETTYFILEINDEX operates by reading expressions from the file and reprettyprinting them to the output image stream, building up an index of the objects as it goes. The index is partitioned by type (e.g. FUNCTIONS, VARIABLES, MACROS, etc.); within each type, the objects are listed alphabetically by name along with the page number(s) on which their definitions appear in the listing.

PRETTYFILEINDEX also modifies the Exec's and the FileBrowser's SEE command to prettyprint the file being viewed, if it is a Lisp source file. It also modifies the PF and PF\* commands to prettyprint the requested function body. Together, these features mean you can use the NEW & FAST options to MAKEFILE to speed up file creation without sacrificing the ability to get pretty listings or see the files prettily inside Lisp.

PRETTYFILEINDEX performs some additional niceties in the listing: it prints bitmaps by "displaying" them, rather than dumping their bits; it translates underscore to left arrow (for the benefit of Interlisp listings); it prints quote and backquote in a font in which they are clearly distinguishable; and it suppresses some of the "noise" in source files, such as the filemap.

The module also contains a function MULTIFILEINDEX that can be used to generate a merged index of items from a whole set of files being listed.

PRETTYFILEINDEX subsumes, and is incompatible with, the modules SINGLEFILEINDEX and PP-CODE-FILE. You can, however, load PRETTYFILEINDEX on top of either one, and it will successfully wrest control of LISTFILES from them. PRETTYFILEINDEX has several advantages over SINGLEFILEINDEX: the prettyprinter has fine control over positioning of the output stream, so things that are supposed to line up do, despite font changes and variable-width fonts; the entire page is used, rather than sacrificing the bottom quarter or so due to lack of control over page breaks; and the use of an image stream allows bitmaps to be rendered directly.

**USING PRETTYFILEINDEX**

For ordinary use, just load PRETTYFILEINDEX.LCOM. This redefines LISTFILES1 so that calling LISTFILES or using the File Browser's Hardcopy command invokes PRETTYFILEINDEX if the file is a Lisp source file. The listing is created by default in a single background process that handles all LISTFILES requests. The file being indexed needn't be loaded, or even noticed (in the File Manager sense) as long as the file's commands don't require customized prettyprinting defined by the file itself. The index is printed at the end of the listing; you are expected to manually transpose the index to the front of the collection of paper that emerges from the printer.

PRETTYFILEINDEX normally assumes that you are printing one-sided listings. However, if your global default is for two-sided (currently this means that EMPRESS#SIDES = 2) or you specified two-sided in the options you passed to LISTFILES, it will prepare the output as if for two-sided listing. For example, from an Interlisp exec,

```
(LISTFILES (SERVER "Perfactor:" %#SIDES 2) FOOBAR)
```

causes the file FOOBAR to be listed two-sided on the print server Perfactor: (the % is the Interlisp reader's escape character, needed to quote the special character #; in an XCL exec the escape character is \, and from other packages you also have to qualify the symbols LISTFILES, SERVER and #SIDES with the package prefix IL:).

For two-sided listings, the margins are symmetric, instead of being shifted a bit to the right, page numbers appear on the outside edge of the page, and a blank page is inserted at the end of the listing if necessary to ensure that the index starts on an odd page (and hence is transposable to the front).

Prettyfileindex prettyprints the file's contents and prints indexed names using the package and read table specified in the file's reader environment, which appears at the beginning of the file. It assumes, as does most of the file manager, that the reader environment is sufficient to read any expression on the file. If you have violated this assumption, for example, by referring in the file to a symbol in another package that is defined on a file that is indirectly loaded by the file somewhere in its coms, you will probably need to LOADFROM the file before you can list it.

### INDEXING MULTIPLE FILES

Ordinarily, you list files and get one index per file. If a module is made up of several files, you may want a master index of the whole set of files, so that you don't have to remember which file contains a function, macro, etc. that you are looking up. This job is handled by MULTIFILEINDEX:

```
(MULTIFILEINDEX files printoptions) [Function]
```

This function lists each of the files in the list *files* using PRETTYFILEINDEX and then produces a master index by merging all the individual indices. The master index is appended to the output of the last file listed. The argument *files* can be a list of file names and/or file patterns, such as "{FS:}<Carstairs>RED\*", or a single such pattern. In the pattern, unless explicitly specified, the extension defaults to null and the version to "highest". The argument *printoptions* is a property-list of options, the same as the *printoptions* argument to SEND.FILE.TO.PRINTER or PRETTYFILEINDEX, with the addition of some options recognized by MULTIFILEINDEX, described further below.

As each file is listed, its pages are numbered with an ordinal file number plus the page number within the file; e.g., in the first file the pages are numbered 1-1, 1-2, ..., in the second file 2-1, 2-2, etc. The master index then refers to page numbers in this form, although each individual file's own index shows only the file-relative page numbers. Alternatively, you can tell MULTIFILEINDEX to number all the pages consecutively, rather than using "part numbers", by giving the option :CONSECUTIVE, value T in *printoptions*.

In the event that some files in the set have different reader environments, the master index is printed in the environment used by the majority of the files. More specifically, MULTIFILEINDEX independently chooses the package used by the majority of the files and the readtable used by the majority; in the case of a tie, the file later in the set wins. If this default is not adequate, you can specify the environment yourself by giving the :ENVIRONMENT option. The value should either be a reader environment object, such as produced by MAKE-READER-ENVIRONMENT, or a property list of the form used by the MAKEFILE-ENVIRONMENT property.

For example,

```
(MULTIFILEINDEX "<Barney>Rub*"
  (:CONSECUTIVE T
   :ENVIRONMENT (:PACKAGE "JABBA" :READTABLE "XCL")))
```

would list each of the files matching "<Barney>Rub\*."; numbering the pages consecutively from the first file through the last, and printing the master index with respect to the package JABBA and read table XCL.

### INCREMENTALLY REPRINTING MULTIPLE FILES

If you have used MULTIFILEINDEX to list a group of files, and later one of the files changes, or maybe the printer just ate part of your listing, you might want to update your listing without reprinting the entire set of files. You have two options.

(1) You can have PRETTYFILEINDEX reprint the one file that changed (or was eaten). Specify the print option :PART *n* to have it treat the single file as the *n*th part of a multiple listing, or the option :FIRSTPAGE *n* to have it start numbering the pages at *n* instead of 1 (for the case where you used the :CONSECUTIVE option to MULTIFILEINDEX). For example,

```
(LISTFILES (:PART 3) "<Barney>Rubric")
```

would reprint <Barney>Rubric as the third file in a group. Of course, this doesn't reprint the master index, but it only has to process the one file, which may be adequate for your needs if things didn't move around too much.

(2) You can have MULTIFILEINDEX process the entire set of files again, but only print some of them. You specify this by parenthesizing the files you don't want printed. That is, each element of the *files* argument to MULTIFILEINDEX is a file name or a list of file name(s); those files inside sublists are processed but not printed. You cannot specify patterns. The master index is listed after the last file, as usual, except that if the last file was in a sublist, and hence not printed, the master index will appear as a separate listing. Calling MULTIFILEINDEX in this manner is nearly as computationally expensive as calling it to list the whole set for real (it omits only the transportation to the printer), but it does save paper and printer time.

### LISTING COMMON LISP FILES

Ordinarily, PRETTYFILEINDEX only processes files produced by the Lisp File Manager; it passes all others off to the default hardcopy routines. However, you can tell it to process a plain Common Lisp text file by passing the print option :COMMON; e.g.,

```
(LISTFILES (:COMMON T) "conjugate.lisp")
```

Prettyfileindex still processes the file by reading and prettyprinting, just as for Lisp files. It starts in the default Common Lisp reading environment (package USER and read table LISP), and evaluates top-level package expressions, such as in-package and import, in order to continue reading correctly. The index is printed in whatever the environment was at the end of the file.

Of course, this is of fairly limited utility, as all read-time conditional syntax is lost: comments, #+, #o, etc. The one exception is that top-level semi-colon comments are preserved—they are copied to the output directly, rather than being read.

## Customizing PRETTYFILEINDEX

The remainder of this document describes various ways in which PRETTYFILEINDEX can be customized.

## HOW TO SPECIFY INDEXING TYPES

Initially, PRETTYFILEINDEX knows about most of the standard file manager types. In addition, it handles all the types defined by DEFDEFINER. For definers with a :NAME option, it assumes that the function is free of side effects. PRETTYFILEINDEX also notices (but does not evaluate) DEFDEFINERs that appear on the file it is currently indexing, which should appear before any instances of the type so defined in order for correct indexing to occur. Of course, it can't know about definer types that are defined on some other file unless you load it.

You can augment the set of indexing types, or override the default handling of definers, by adding elements to the following variable:

**\*PFI-TYPES\*** [Variable]

A list of entries describing types to be indexed and a way of testing whether an expression on the file is of the desired type. Each entry is a list of up to 4 elements of the form (*type dumpfn namefn ambiguous*), the first two of which are required:

- type* The name of the type, e.g., MACRO. This name will appear as the name of the index for this type, e.g., "MACRO INDEX". *type* is usually the name of a file package type, though it need not be. It must be a symbol.
- dumpfn* The name of the function that appears as the CAR of the form that defines objects of type *type* on the file, or a list of such names. E.g., for type TEMPLATE it is SETTEMPLATE; for type VARIABLES it is (RPAQ RPAQQ RPAQ? ADDTOVAR).
- namefn* A function that tests whether the expression that starts with *dumpfn* really is of the desired type, and returns the name of the object defined in the expression. The function takes as arguments (*expr entry*), where *expr* is the expression whose CAR matched the entry. The *testfn* should return one of the following:
  - NIL the expression is not of the desired type.
  - name* the expression defines a single object of this name and of the type given in the entry.
  - a list* the value is either a single list or a list of lists, each of the form (*type . names*), meaning that the expression defines each of the names as having the specified type.

If the *namefn* is NIL or omitted, the name of the object is obtained from the second element of the expression. If that element is a list, the name is taken to be its CAR, or its CADR if the element is a quoted atom.

- ambiguous* True if the expression is ambiguous, in the sense that even if *namefn* returns a non-NIL value, it is possible for this expression to also satisfy other entries in \*PFI-TYPES\*. E.g., the expression (RPAQ --) is ambiguous, because it could define either a variable or a constant. If *ambiguous* is true, you usually want a corresponding entry on \*PFI-FILTERS\* (below).



**\*PFI-PROPERTIES\***

[Variable]

A list used by the default handler for the PUTPROPS form. It associates property names with a type (something more specific than the type PROPERTY) under which objects having this property should be indexed. Each element is of the form (*propname type*). If *type* is NIL or omitted, then objects having this property are ignored. In addition, the default PUTPROPS handler treats all elements of the list MACROPROPS as implying type MACRO.

The initial value of **\*PFI-PROPERTIES\*** is

```
((COPYRIGHT)
 (READVICE ADVICE)),
```

meaning that the COPYRIGHT property should be ignored, and the READVICE property implies that the object should be indexed as type ADVICE.

**\*PFI-FILTERS\***

[Variable]

A list describing potential index entries that should be filtered out of the final index. Each element of **\*PFI-FILTERS\*** is a list (*type filterfn*), where *type* is one of the types in **\*PFI-TYPES\*** and *filterfn* is a function of one argument, an index entry. If *filterfn* returns true, then the index entry is discarded. An index entry is of the form (*name pagenumbers*). For convenience, an element of **\*PFI-FILTERS\*** can also take the form (*type . subtype*), meaning that if an object is already indexed as a *subtype* then it should not also be indexed as a *type*.

The initial value of **\*PFI-FILTERS\*** is

```
((VARIABLES . CONSTANTS)),
```

meaning that "variables" that successfully index as constants should not also be listed in the VARIABLES index. This extra pass is needed because the CONSTANTS File Manager command causes expressions of the form (*RPAQ var value*) to be dumped on the file, and at the time this expression is read, it is not known whether there will later on appear a CONSTANTS form for the same variable.

Filter functions may want to call the following function:

```
(PFI.LOOKUP.NAME name type)
```

[Function]

Looks up *name* in the index being built for type *type*. If it finds an entry, it returns it. Index entries are of the form (*name . pagenumbers*). It is permissible for a filter function as a side effect to destructively change another index entry by adding page numbers to it. You might want to do so, for example, in the case where there is a kind of object that dumps two expressions on a file, each of which is a different type (according to **\*PFI-TYPES\***), but you want both occurrences indexed as a single type.

**MORE EXPLICIT EXPRESSION HANDLING**

The functions and variables described below allow you to completely control how certain expressions in the input file are handled. You can use these hooks to perform custom prettyprinting, to suppress the printing of some expressions, or to perform indexing more complex than that supported by **\*PFI-TYPES\***.

**\*PFI-HANDLERS\***

[Variable]

An association list specifying explicit "handlers" for expressions that appear on the input file. Each element is a pair (*car-of-form* . *handler*), where *handler* is a function of one argument, an expression read from the file whose first element is *car-of-form*. The handler is completely in charge of indexing the expression and/or printing it to \*STANDARD-OUTPUT\*. Unless the handler chooses to suppress the printing altogether, it is expected to print at least one blank line first, so that expressions are attractively separated in the listing (see PFI.MAYBE.NEW.PAGE).

**\*PFI-PREVIEWERS\***

[Variable]

This list is used when PRETTYFILEINDEX is used by the SEE command. During the SEE command, real-time performance is important, so it is undesirable to have long delays while reading a very large expression. For example, all the functions in an Interlisp FNS command appear on the file inside a single DEFINEQ expression. If handled in the obvious way, the user would have to wait for the entire expression to be read before any output appeared. A previewer has the opportunity to read the expression in pieces and prettyprint it as it goes.

Each element of \*PFI-PREVIEWERS\* is a pair (*car-of-form* . *previewer*), where *previewer* is a function of one argument, the *car-of-form*. The previewer is called when PRETTYFILEINDEX encounters an expression of the form "(*car-of-form* " on the file. Its job is to read expressions from \*STANDARD-INPUT\* (currently positioned after the *car of form*) until it encounters the closing right parenthesis, which it should consume, and prettyprint the elements appropriately to \*STANDARD-OUTPUT\*. \*PFI-PREVIEWERS\* is used only from the SEE command, so indexing is not necessary (but also not harmful, other than to waste some time).

If an expression does not have a previewer, PRETTYFILEINDEX reads the rest of the expression itself and handles it normally, i.e., performs (PFI.HANDLE.EXPR (CONS *car-of-form* (CL:READ-DELIMITED-LIST #\))).

(PFI.DEFAULT.HANDLER *expr*)

[Function]

This is the function PRETTYFILEINDEX uses to process expressions that have no explicit handler. It indexes the expression according to \*PFI-TYPES\* and then prettyprints the expression. You can call this function from your handler if you decide you have an expression you didn't want to handle specially.

(PFI.HANDLE.EXPR *expr*)

[Function]

Performs PRETTYFILEINDEX's normal handling of the expression *expr*, including looking on \*PFI-HANDLERS\*. Handlers and previewers of forms that encapsulate arbitrary expressions, such as DECLARE :, typically call this to process subexpressions.

(PFI.ADD.TO.INDEX *name type/entry*)

[Function]

Adds an entry to the index for *type/entry* specifying that *name* occurs on the current page. *type/entry* is either a type or an entry from \*PFI-TYPES\* from which the type will be extracted.

(PFI.PRETTYPRINT *expr name formflg*) [Function]

Prettyprints *expr*. Optional *name* is the name of the object being printed; if a page crossing occurs in the middle of the prettyprinting, this name will be displayed in the page header. If *formflg* is true, print the expression as code; otherwise as data.

(PFI.MAYBE.NEW.PAGE *expr minlines*) [Function]

Starts a new page if the listing is currently near the bottom of the page and *expr* won't fit, else performs a single (TERPRI). If *minlines* is specified, it is an explicit estimate of how much space the expression will require, in which case *expr* can be NIL; otherwise, the function estimates the size. Handlers should call this *before* calling PFI.ADD.TO.INDEX, so that the page number in the index is correct. The typical handler calls PFI.MAYBE.NEW.PAGE, then PFI.ADD.TO.INDEX, then prints the expression, possibly via PFI.PRETTYPRINT.

## OTHER VARIABLES

\*PFI-INDEX-ORDER\* [Variable]

A list of types (as in \*PFI-TYPES\*) in the order in which the various types should appear in the index. Types not in this list are printed in an order of the program's choosing, currently a "best fit" algorithm (print the largest type index that will fit on the page). The initial value is (FUNCTIONS), meaning that the function index will appear first, with no constraints on the order of other types.

\*PFI-PRINTOPTIONS\* [Variable]

A list of print options that PRETTYFILEINDEX appends to the list of print options passed to LISTFILES, thus supplying some printing defaults. The initial value is (REGION (2540 1905 17780 24765)) which on standard letter size paper in portrait mode results in left, bottom, top, and right margins of 1",  $\frac{3}{4}$ ",  $\frac{1}{2}$ " and  $\frac{1}{2}$ ", respectively. If the print options passed to LISTFILES call for a two-sided listing, the default region is shifted  $\frac{1}{4}$ " to the left. If the print options specify LANDSCAPE mode, the default region is ignored.

\*PFI-MAX-WASTED-LINES\* [Variable]

If an expression looks like it won't fit on the current page and there are no more than this many lines remaining on the page, PRETTYFILEINDEX starts a new page before printing the expression. A floating-point value indicates a fraction of the page; an integer indicates an absolute number of lines. The initial value is 12.

\*PFI-CHARACTER-TRANSLATIONS\* [Variable]

A list specifying how certain characters should be rendered on the output stream. This is used to get around the poor rendering of certain characters in the default font. Each element is of the form (*imagetype* . *charpairs*), where *imagetype* is the type of image stream being printed to and each element of *charpairs* is an alist whose elements are of the form (*sourcecode* *destcode* . *looks-plist*), specifying the character code to use on the destination image stream for a specified character code in the input stream. If *looks-plist* is non-NIL, *destcode* is printed in a font obtained by applying FONTCOPY to the current font and *looks-plist*.

The initial value is

```
((INTERPRESS (95 172)
              (96 169 FAMILY CLASSIC)
              (39 185 FAMILY CLASSIC)))
```

meaning if the output stream is an Interpress stream the lister should turn character 95 (underscore) into 172 (left arrow), backquote into left single quote in the Classic font (of the same size and weight), and single quote into right single quote in Classic.

**\*PRINT-PRETTY-FROM-FILES\*** [Variable]

If true, the SEE (in the Exec and Filebrowser), PF and PF\* commands attempt to prettyprint to the display, rather than copying the file as it is currently formatted. The initial value is T.

**\*PRINT-PRETTY-BITMAPS\*** [Variable]

If true, then when **\*PRINT-ARRAY\*** is true and a bitmap is to be printed to an image stream, the bitmap itself is displayed as an image on the stream, rather than as the machine-readable representation of its bits (of the form `##(16 16)H@@@L.`). This variable has no effect on printing to files, such as in MAKEFILE, nor on PRETTYFILEINDEX, which binds it true; thus, changing the value mainly affects the display. The initial value is T.

**\*PFI-DONT-SPAWN\*** [Variable]

If NIL, LISTFILES arranges for a separate process to do the hardcopying (whether using PRETTYFILEINDEX or not) and returns immediately; if T, it makes the listing directly, not returning until it is finished. The initial value is NIL.

### LISTING ELSEWHERE THAN THE PRINTER

Ordinarily, you call LISTFILES (or uses the File Browser) to create listings. However, you can also call PRETTYFILEINDEX directly if you want to direct the output elsewhere, such as to an Interpress file:

(PRETTYFILEINDEX *filename printoptions outstream dontindex*) [Function]

Lists *filename*, the name of a Lisp source file or a stream open for input on such a file, printing it and its index to *outstream*. *outstream* is either an open image stream, or NIL, in which case the output goes to (OPENIMAGESTREAM) and the stream is closed afterwards, which results in it being sent to the default printer. If *filename* or *outstream* is open on entry, it is left open on exit. *printoptions* is a plist of options of interest to either LISTFILES or OPENIMAGESTREAM. If *dontindex* is true, no index is produced; this argument is used by the SEE command.

If the file is not a File manager file, PRETTYFILEINDEX takes no action and returns NIL; otherwise, it returns the full file name. However, if *filename* is an open stream, then PRETTYFILEINDEX copies the remainder of the stream to *outstream* (which must be given) using PFCOPYBYTES, and returns the full file name. This is so that the stream does not need to be backed up after discovering that the file is not a File Manager file, an operation not possible for a sequential-access stream.

### LIMITATIONS

Prettyfileindex assumes that the default font, which is used to print the index, is fixed-width.

PRETTYFILEINDEX uses the regular Interlisp prettyprinter. This means that if you have File Manager commands that produce their output in a customized way, e.g., by printing inside the E command, then the output will look different between MAKEFILE and PRETTYFILEINDEX. You can usually remedy this by supplying PRETTYPRINTMACROS for the types of expressions your command dumps (which may also let you replace the E with a simpler P command), or by defining handlers for the expressions (see \*PFI-HANDLERS\*). PRETTYFILEINDEX already supplies PRETTYPRINTMACROS for most of the customized printing done by the current File Manager: RPAQ, RPAQQ, RPAQ?, ADDTOVAR, PUTPROPS and COURIERPROGRAM.

With the exception of noticing the reader environment and DEFDEFINER expressions, PRETTYFILEINDEX does not interpret the contents of the file. If your file depends on itself for proper prettyprinting or indexing, you need to LOAD (or possibly just LOADFROM) the file first.

---

---

**PRINTERMENU**

---

---

By: ISLWS (Bloomberg.pa@Xerox.com)

**DESCRIPTION**

Creates a menu which displays all printers in the global list DEFAULTPRINTINGHOST, allows selection of a default printer, and permits addition and deletion of printers from DEFAULTPRINTINGHOST. Printers are displayed in the same order as they appear in DEFAULTPRINTINGHOST. Selecting an item from the menu will highlight by inversion and move it to the top of the menu, thus becoming the default printer. Selection in the title bar of the menu with the left or middle button will allow you to add or to delete a printer from the menu.

An auxiliary process, PRINTERMENU.WATCH, monitors the value of DEFAULTPRINTINGHOST and will update the menu if this variable is changed. If PRINTERMENU.WATCH is killed, the menu will be grayed out to indicate that it may no longer be valid. Clicking left or middle buttons inside the menu will restart PRINTERMENU.WATCH and update the menu.

**To use:**

Load the module with:

(LOAD 'PRINTERMENU.LCOM)

Start it with:

(PRINTERMENU)

**User Switches:**

1. Set DEFAULTPRINTINGHOST to contain all printers from which you wish to select.
2. Prior to calling the function (PRINTERMENU), the global variable PRINTERMENU.POSITION can be set to the position, in screen coordinates, where you want the menu to appear. If not set, you will be prompted for a position for the menu window.

---

---

**PROGRAMCHAT**

---

---

By: ISLWS (bloomberg.pa@xerox.com)

**DESCRIPTION**

PROGRAMCHAT is a Lisp function that invokes a windowless Chat process to execute a single command line on a remote host. PROGRAMCHAT requests a *login* if one has not been made recently to the remote host. After execution of the command, a normal *logout* is performed, and the Chat connection is closed.

PROGRAMCHAT was written by Eric Schoen to allow initiation of remote computation from Lisp workstations. It works with both VMS and Unix operating systems on the remote host.

**To use:**

Load the module with:

```
(LOAD 'PROGRAMCHAT.LCOM)
```

Invoke the function with:

```
(PROGRAMCHAT hostname commandString windowFlg) [Function]
```

where

*hostname* is the network name of the remote host,

*commandString* is a string which is the exact format of the command to be run from the command line interpreter of a VAX/VMS host (or from the shell of a VAX/Unix host), and

*windowFlg* is a variable that, when T, opens a window and displays a log of data transferred between the PROGRAMCHAT process and the remote host. PROGRAMCHAT is normally invoked with *windowFlg* = NIL.

**Warnings:**

1. When loaded, PROGRAMCHAT resets the variable NETWORKLOGINFO.
2. PROGRAMCHAT provides no error handling. If the connection to the remote host is broken, no error message is returned.

---



---

## PROMPTREMINDERS

---



---

To be periodically reminded of things

By: JonL White

Revised by: Larry Masinter (Masinter.pa@Xerox.com), subsequently by Becky Burwell  
(Burwell.PA@Xerox.COM)

### INTRODUCTION

PROMPTREMINDERS implements a facility which schedules events to be performed, or messages to be flashed in a prompt window. Events can be periodic or once-only. The showing of a message in a prompt window has the extra facility of flashing a message, and stopping only when there has been a recent response (mouse or keyboard movement) from the user.

If the MESSAGE given for the reminder (see description of the function SETREMINDER below) is a listp, then when the reminder "goes off", that listp will be EVAL'd rather than any of the "winking", "flashing", or "hassling" mentioned above.

The global variable REMINDERSTREAM holds the stream where the message is to be displayed; if not set by the user, it defaults to PROMPTWINDOW. After the message has been displayed, the window (if indeed REMINDERSTREAM holds a window) will be closed, depending on the value of CLOSEREMINDERSTREAMFLG.

REMINDERS is a file package type, so that they may be easily saved on files, and so that the general typed-definition facilities may be used. On any file which uses the REMINDERS filepkgcom, it is advisable to precede this command with a command

```
(FILES (SYSLOAD COMPILED FROM LISPUSERS) PROMPTREMINDERS)
```

since this package is not in the initial Lisp loadup. When initially defining a reminder, it is preferable for the user to call SETREMINDER rather than PUTDEF; but HASDEF is the accepted way to ask if some name currently defines a "reminder", and DELDEF is the accepted way to cancel an existing "reminder".

### EXAMPLES

```
(SETREMINDER NIL (ITIMES 30 60) "Have you done a CLEANUP recently?")
```

the user wants to be reminded every 30 minutes that he ought to save his work

```
(SETREMINDER 'WOOF NIL "Call home about dinner plans."  
"8-Jan-83 4:00PM")
```

he merely wants to be told once, at precisely 4:00PM to call home



```
(SETREMINDER NIL 600
  '(PROGN (AND (FIND.PROCESS 'LISTFILES) (add FREQ 1))
    (add TOTAL 1)))
```

checks every 10 minutes to see if there is a process called LISTFILES

## FUNCTIONS

(SETREMINDER NAME *PERIOD MESSAGE INITIALDELAY EXPIRATION*) [Function]

This will create and install a "reminder" with the name NAME (NIL given for a name will be replaced by a gensym), which will be executed every PERIOD number of seconds by winking the string MESSAGE into the prompt window; if MESSAGE is null, then NAME is winked; if MESSAGE is a listp, then it is EVAL'd and no "winking" takes place. "Winking" means alternately printing the message and clearing the window in which it was printed, at a rate designed to attract the eye's attention.

The first such execution will occur at PERIOD seconds after the call to SETREMINDER unless INITIALDELAY is non-NIL, in which case that time will be used; a numeric value for INITIALDELAY is interpreted as an offset in seconds from the time of the call to SETREMINDER, and a stringp value is an absolute date/time string.

If PERIOD is null, then the reminder is to be run precisely once. If EXPIRATION is non-null, then a fixp means that that number of seconds after the first execution, the timer will be deleted; a stringp means a precise date/time at which to delete the timer.

Optional 6th and 7th arguments -- called REMINDINGDURATION and WINKINGDURATION -- permit one to vary the amount of time spent in one cycle of the wink/flash loop, and the amount of time spent winking before initiating a "flash". The attention-attracting action will continue for REMINDINGDURATION seconds (default: the value of the global variable DEFAULT.REMINDER.DURATION which is initialized to 60), or until some keyboard action takes place.

Type-ahead does not release the winking. In case the user fails to notice the winking, then every WINKINGDURATION seconds (default: the value of the global variable DEFAULT.REMINDER.WINKINGDURATION which is initialized to 10) during the "reminding", the whole display videocolor will be wagged back and forth a few times, which effects a most obnoxious stimulus.

SETREMINDER returns the name (note above when NIL is supplied for the name).

(ACTIVEREMINDERNAMES) [Function]

ACTIVEREMINDERNAMES returns the list of active reminders.

(REMINDER.NEXTREMINDDATE NAME DATE) [Function]

REMINDER.NEXTREMINDDATE returns (and optionally sets) the date&time (in DATE format) at which the reminder is next to be executed.

(REMINDER.EXPIRATIONDATE NAME DATE) [Function]

REMINDER.EXPIRATIONDATE returns (and optionally sets) the date&time (in DATE format) at which the reminder will be automatically deleted.

(REMINDER.PERIOD *NAME SECONDS*)

[Function]

REMINDER.PERIOD returns (and optionally sets) the period (in seconds) at which the reminder gets rescheduled.

(SHOWDEF *name* 'REMINDERS')

[Function]

will show a reminder in a pretty format, etc.

---

---

## Proofreader

---

---

By: John Maxwell (Maxwell.pa@Xerox.com)

Use in conjunction with Analyzer, SpellingArray

### INTRODUCTION

The Proofreader interactively looks for and corrects spelling errors in a given TEdit document. To use it, go to the TEdit menu (click the middle button while in the title bar) and invoke the menu item labeled "Proofread". This will simultaneously attach a special menu to the side of the document and start proofreading from the caret. The proofreader scans the document from the caret location and stops at the first misspelling it finds, highlighting it with a pending delete selection. Successive misspellings can be found by clicking the "Proofread" menu item in the TEdit menu or the "Proofread" menu item in the new side menu.

At this point, you can either correct the misspelled word or skip to the next misspelling. If you are not sure what the correct spelling is, you can get a menu of possible corrections by invoking the "Correct" menu item. Selecting a correction from this menu will cause it to be automatically inserted into the document in place of the misspelling. (Note: The Proofreader occasionally suggests some very bizzare spelling corrections for your misspelled word. Do not be alarmed; this is a known but unavoidable artifact of the heuristic used for checking for misspellings. (see notes at the end)) If the word was erroneously flagged as misspelled, then you can insert the word into your personal word list of acceptable words by invoking the "Insert" menu item. At the end of the editing session you can save the word list on a file with the "StoreWordList" command (see below).

If when the Proofreader is invoked the current selection has more than one word in it, then the Proofreader will only correct the words in that selection. Otherwise, the Proofreader always proofreads the text from the caret to the end of the document.

### PROOFREADER SUB-COMMANDS

Under the TEdit Proofread menu item there are a number of sub-commands that the user can invoke. They are:

#### **Proofread**

The same as the top level Proofread command. Attaches a special menu to the side of the document and starts proofreading from the caret.

#### **CountWords**

Counts the number of words in the current selection. To count the number of words in the entire document, first click "All" in TEdit's expanded menu.

**SetProofreader**

Gives the user a menu of proofreaders to use for proofreading. If there is only one proofreader, no menu is generated. If the user selects a remote server, a second menu may be generated of proofreaders available on the server.

**StoreWordList**

Allows the user to save the words that he inserted into the proofreader onto a remote file. The words from an existing version of the file will be read in first and then a new file will be generated consisting of the old file plus the new words. After the file is written, the list of newly inserted words is set to NIL.

**LoadWordList**

The inverse of **StoreWordList**. If you have a file that you want to load every time you use the proofreader, you can add it to `Proofreader.AutoLoad`, and it will be loaded when the proofreader is first opened. `Proofreader.AutoLoad` can be either a file or a list of files.

**AutoCorrect**

Sets the variable `Proofreader.AutoCorrect` so that the proofreader *automatically* generates a list of corrections whenever it finds a misspelled word. In the `AutoCorrect` mode, the proofreader will also automatically scan for the next misspelled word whenever after a correction has been selected. If you want to stop the proofreading process, click outside of the correction menu. If you want to insert the flagged word into your word list, click the menu item labeled `"*INSERT*"`. If you want to continue proofreading without changing the flagged word, select the menu item labeled `"*SKIP*"`.

**ManualCorrect**

Sets the variable `Proofreader.AutoCorrect` so that the proofreader will not generate a list of corrections unless the user asks for it.

**PROOFREADER VARIABLES**

There are a couple of variables that the user can set in his init file to change how the proofreader works. They are:

`Proofreader.AutoLoad` [Variable]

A file or list of files to be loaded into the proofreader every time that the proofreader is initialized.

`Proofreader.AutoCorrect` [Variable]

A boolean that determines whether or not a list of corrections is automatically generated whenever the proofreader finds a misspelled word. The default value is NIL.

`Proofreader.AutoDelete` [Variable]

A boolean that determines whether or not to delete the old versions of a word list file when a new one is written out. The default value is T.

`Proofreader.MenuEdge` [Variable]

The side of the window that the proofreader menu appears on (can be either LEFT or RIGHT). The default value is LEFT.

Proofreader.UserFns

[Variable]

A list of functions to be applied to misspelled words (as strings). If the function returns a non-NIL value, then the word is assumed to be correctly spelled. (If the function ATOMHASH#PROBES is added to Proofreader.UserFns, then any word defined as an atom becomes legal. ATOMHASH#PROBES tests whether or not a string is an atom without creating new atoms. If you want a more restricted test (i.e. "anything defined as a procedure is legal") first test that the string exists at an atom before doing MKATOM. Otherwise, the atom space will fill up with misspelled words.)

#### NOTES

- The proofreader uses a heuristic to determine whether or not a word is in its word list that occasionally will produce false positives. This is most noticeable when the proofreader is generating corrections for a misspelled words. I don't know of any way to eliminate this problem except to use a different algorithm, the fastest of which is at least twice as slow. Hopefully people will find it more of a nuisance than a real problem.
- There is no way to remove words once they have been inserted into the local dictionary. The only way to get rid of a bad word is to reload the dictionary. This can be done by reloading the SpellingArray file. If a bad word gets into one of the remote files, you can edit the file to get rid of it.

#### ACKNOWLEDGMENTS

The algorithm used in the Proofreader is based on the algorithm in the Cedar Spelling Tool by Bob Nix. For more information on the implementation, see the section "How it Works" in {Cyan}<CedarChest6.1>Documentation> SpellingToolDoc.tioga.

---



---

## QEdit

---



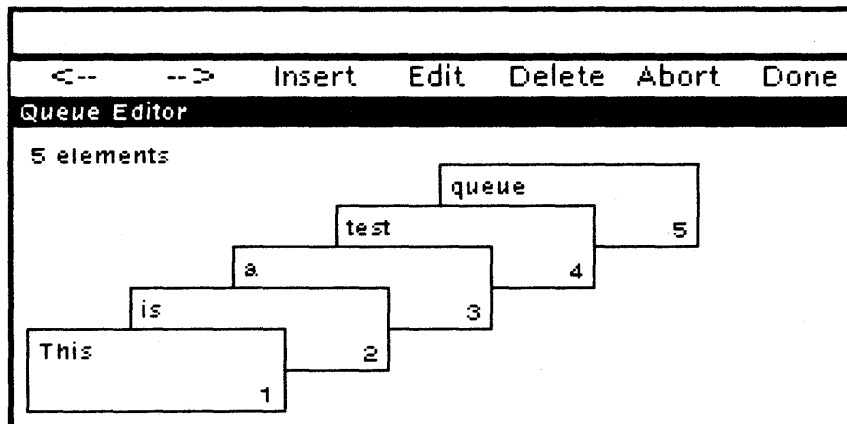
---

By: Johannes A. G. M. Koomen  
(Koomen.wbst@Xerox or Koomen@CS.Rochester.edu)

This document last edited on January 22, 1988

### INTRODUCTION

QEdit is a facility for editing queues, which are ordered lists containing arbitrary objects. QEdit provides facilities for reordering, editing and deleting current queue entries, and inserting new entries. A QEditor looks like this:



Clicking the left mouse button over a queue entry makes it the current selection. You can then select a QEdit menu operation. QEdit returns with the original list if aborted, or a new list upon normal exit. Selecting an entry that is not entirely visible causes the display to scroll until it is.

### DETAILS

**(QEDIT QUEUE PROPS)**

[Function]

Invokes a QEditor on *QUEUE*. Returns *QUEUE* if aborted, a new list otherwise. The *PROPS* argument is a property list which serves to customize the behavior of QEDIT for this particular invocation. Currently defined props are listed below.

<—

[QEdit command]

Moves the current selection forward, i.e., switches the current selection and the queue entry just before it.

—>

[QEdit command]

Moves the current selection backward, i.e., switches the current selection and the queue entry just after it.

Insert	[QEdit command]
Inserts a new entry in front of the current selection, provided <i>PROPS</i> contained an <i>INSERTFN</i> .	
Edit	[QEdit command]
Edits the current selection, provided <i>PROPS</i> contained an <i>EDITFN</i> .	
Delete	[QEdit command]
Deletes the current selection, provided <i>PROPS</i> contained a <i>DELETFN</i> .	
Abort	[QEdit command]
Aborts the current QEdit session, returning the original queue.	
Done	[QEdit command]
Ends the current QEdit session, returning a new list containing the current queue entries.	
TITLE	[QEdit property]
If supplied, its value becomes the title for the QEdit window.	
CONTEXT	[QEdit property]
The value of the <i>CONTEXT</i> property is passed to the functions below as an extra argument. It does not affect QEdit directly. The functions below can also obtain the current queue by calling the function <i>QEDIT.CURRENT.QUEUE</i> (see below).	
LABELFN	[QEdit property]
If supplied, its value is a function which, when invoked on a queue entry and the user context, returns a label to use for displaying the queue entry. If not supplied, QEdit displays the queue entry itself.	
LABELFONT	[QEdit property]
If supplied, its value is a font specification for displaying the queue entry.	
INSERTFN	[QEdit property]
If supplied, its value is a function which, when invoked on the user context, returns either <i>NIL</i> or a new element to be inserted in front of the current selection (at the front of the queue, if there is no current selection). If not supplied, no elements can be inserted into the queue.	
EDITFN	[QEdit property]
If supplied, its value is a function which, when invoked on a queue entry and the user context, returns either <i>NIL</i> or a (possibly new) entry to be used instead of the current selection.	
DELETFN	[QEdit property]
If supplied, its value is a function which, when invoked on a queue entry and the user context, returns <i>NIL</i> if the entry should not be deleted. If not supplied, no elements can be deleted. Hint: the function <i>TRUE</i> always returns <i>T</i> .	
( <i>QEDIT.CURRENT.QUEUE</i> )	[Function]
Invoked from one of the above mentioned functions, returns the queue being edited in its current form. The <i>INSERTFN</i> might use this, for example, if duplicates are not allowed.	

(QEDIT.RESET) [Function]

QEdit will reuse QEditors upon reinvocation. This function will throw away any known but currently inactive QEditors. Useful if you wish to change the default QEdit props.

\*QEDITPROPS\* [QEdit variable]

Any props not explicitly supplied in the call to the function QEDIT are taken from this free variable. Its initial value is (TITLE "Queue Editor" LABELFONT (HELVETICA 8))

#### Examples

(QEDIT '(This is a test queue)) [Function]

Brings up a QEditor as shown above. Queue elements can be rearranged, but not added to, edited or deleted.

(QEDIT '(This is a test queue) '(INSERTFN READ DELETEDFN TRUE)) [Function]

Brings up a QEditor as shown above. Queue elements can be rearranged, inserted or deleted, but not edited.



---



---

## READAIS

---



---

By: Nick Briggs (Briggs.pa@Xerox.com)

### INTRODUCTION

AIS (array of intensity samples) is a format for color and gray-level images. The following functions allow reading and writing of AIS files from Lisp.

*(AISBLT FILE SOURCELEFT SOURCEBOTTOM DESTINATION DESTLEFT DESTBOTTOM WIDTH HEIGHT HOW FILTER NBITS LOBITADDRESS)* [Function]

Puts the image in an AIS file into a bitmap. AISBLT checks the sample size of the AIS file and the number of bits per pixel of the DESTINATION and performs the required reduction (if any). SOURCELEFT and SOURCEBOTTOM give the left and bottom coordinates in the source file of the image to be read (default to (0,0)). DESTINATION can be a bitmap, a color bitmap, or a window.

HOW indicates what method of reduction is to be used if the sample size of the AIS file is larger than the number of bits per pixel in the destination bitmap. The recognized methods are TRUNCATE (use the high-order bits) and FSA (use the Floyd-Steinberg dithering algorithm). The default when going to a 1 bpp bit map is FSA; otherwise it is TRUNCATE.

FILTER if non-NIL should be an array that will be used to filter the samples read from the AIS file. If FILTER is given and a sample point of intensity N is read from the file, (ELT FILTER N) is used to determine the bits for the destination. The function SMOOTH HIST described below is one way of getting a filter that balances the contrast in an image.

NBITS and LOBITADDRESS allow an image to be read into one or more "planes" of a color bitmap. NBITS tells how many bits are to be taken from each image sample, and LOBITADDRESS indicates the lowest bit within each pixel that the NBITS bits are to go. (Bit address zero is the leftmost or highest-order bit. For a four-bit-per-pixel bit map, three would be the lowest-order bit.) This is used by SHOWCOLORAIS to put the different planes of a color image into the bit map.

*(SHOWCOLORAIS BASEFILE COLORMAPINFO HOW SOURCELEFT SOURCEBOTTOM DESTINATION DESTLEFT DESTBOTTOM WIDTH HEIGHT)* [Function]

Reads a color image from three AIS files into a color bit map. The three color files are obtained by concatenating the strings "-RED.AIS", "-GREEN.AIS", and "-BLUE.AIS" onto the end of BASEFILE. If COLORMAPINFO is a list of three small integers, it indicates how many of the bits in the destination are allocated to each color. For example, if DESTINATION is a four-bit-per-pixel color bit map and COLORMAPINFO is (1 2 1), one bit (bit zero) will be allocated to the red image, two bits (bits one and two) will be allocated to the green image, and one bit (bit three) will be allocated to the blue image.

DESTINATION is the color bitmap the image will be stored into.

HOW, SOURCELEFT, SOURCEBOTTOM, DESTLEFT, DESTBOTTOM, WIDTH, and HEIGHT are as described in AISBLT.

An experimental feature that is available only when going to 8 bpp color bit map: if COLORMAPINFO is a color map, each pixel will be determined by finding the color in the color map

that is closest to the 24 bits of color information read from the three image files. (This takes a long time.) The function COLOR.DISTANCE (red green blue redentry greenentry blueentry) is called to calculate the distance by which "closest" color is determined.

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

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

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

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

**(GRAYCOLORMAP BITSPERPIXEL)** [Function]

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

**(WRITEAIS BITMAP FILE REGION)** [Function]

Writes the region REGION of the color bit map BITMAP onto the file FILE in AIS format. This provides an efficient way of saving color or gray- level images.

**(AISHISTOGRAM FILE REGION)** [Function]

Returns a histogram array of the region REGION in the AIS file FILE. The histogram array has as its Nth element the number of pixels in the region that have intensity N.

**(GRAPHAISHISTOGRAM HISTOGRAM W)** [Function]

Draws a graph of a histogram array in the window W. If W is NIL, a window is created.

**(SMOOTHEDFILTER HISTOGRAM)** [Function]

Returns a "filter" array that maximally distributes the intensities values contained in HISTOGRAM. The filter array can be passed to AISBLT to change the contrast of the image being read.

---



---

## READAPPLEFONT

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: READDISPLAYFONT

READAPPLEFONT defines a new display font type which allows Envos Lisp to use (commercially available) Macintosh™ display fonts. Although this module is primarily intended for extracting font width information for conversion programs (eg. MacPaint™ to Sketch and vice versa) it can also be used to extend the number of display fonts available within the Envos Lisp environment.

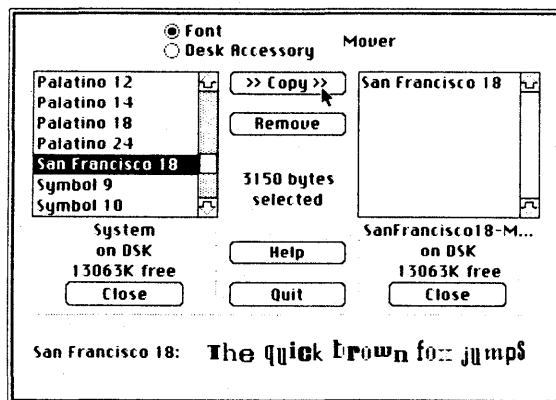
There are *no* user functions in the module. In addition to this module, you need a directory of extracted Macintosh™ font files (as described below) and you must add the name of that directory to the system DISPLAYFONTDIRECTORIES list. The following function is added under the type APPLE to the list DISPLAYFONTTYPES (defined by the READDISPLAYFONT module).

(READAPPLEFONT STREAM FAMILY SIZE FACE) [Function]

The module also adds the extension *APPLE* to the system list DISPLAYFONTEXTENSIONS.

### FONT FILES

This module only uses the FontRec portion of the font files (see *Inside Macintosh™*). The font resources must be extracted into individual files with appropriate names, eg. SANFRANCISCO18-MRR-C0.APPLE. One method of doing this is to use the *Font/DA Mover™* utility:



The individual font files should be moved to a Unix AUFS/CAP server (or the equivalent) and the resource forks (in the .resource directory) should be copied to the directory you added to DISPLAYFONTDIRECTORIES above.

If another method for extracting the FontRec data structure into individual files is used, the following variable will probably need to be reset:

APPLEFONTREC.OFFSET [Variable]

The offset in bytes into the font file of where the FontRec data structure begins (initially 264).

### NOTES

- This module only handles proportionally spaced fonts and ignores fractional character widths.
- The user is responsible for determining the legality of extracting the fonts in question.

---

---

**READBRUSH**

---

---

By: Larry Masinter (Masinter.pa@Xerox.com)

uses: BITMAPFNS

This document last edited on September 8, 1988.

**INTRODUCTION**

This module implements two things:

(IDLE.GLIDING.BRUSH W box wait) [Function]

Like the default IDLE.BOUNCING.BOX Idle function but glides the bitmap around the screen instead of bouncing it.

(READBRUSHFILE file) [Function]

Reads files in the ".brush" format used by Mesa/Viewpoint. Returns a pair of bitmap/mask (or just (bitmap) if there is no mask. Brush file names use defaults from BRUSHDIRECTORY, initially {goofy:osbu north:xerox}<hacks>data>brushes> (and of use only inside Xerox.)

(READBRUSH file) [Function]

Calls READBRUSHFILE and then creates a window with that brush in it.

BRUSHDIRECTORY [Variable]

Default location to get brushes from.

Adds an entry to IDLE.FUNCTIONS for "Gliding box", which will use IDLE.GLIDING.BOX on the brush selected from a menu created by enumerating all of the .brush files on BRUSHDIRECTORY.

---

---

**READDATATYPE**

---

---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

READDATATYPE gives @ a read macro definition (in the INTERLISP readtable) so that it can be used to type in datatype pointers directly. For example, suppose you have lost your pointer to a window (or menu, etc.) but you have the printed representation around (eg. {WINDOW}#56,17470) then you can do things like:

```
90 (INVERTW @{WINDOW}#56,17470)
{WINDOW}#56,17470
```

The read macro is only intended to be used at the *read-eval-print* loop. If the character following the @ is not a { then the read macro returns the @ character just as if you had typed it in so that other expressions that use @, like '(A B ,@FILELST C D), will still work correctly.

Although the read macro does not need the data type name in the brackets (eg. {MENU}) to get the pointer, it does require it in order to check the pointer to make sure it is of the correct *type*. If the pointer is not of the type specified, then the read macro returns NIL.

The following form is used in the COMS of the file to set the syntax of @ in the INTERLISP readtable and can be used to add the capability to other readtables and/or characters:

```
(SETSYNTAX '%@ '(MACRO FIRST READDATATYPE) (FIND-READTABLE "INTERLISP"))
```

---

---

**READDISPLAYFONT**

---

---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

READDISPLAYFONT modifies the display font functions to make it possible to define new display font types.

The functions \READDISPLAYFONTFILE and FONTFILEFORMAT are modified to use the list:

DISPLAYFONTTYPES [Variable]

An ALST containing font file extensions and the functions that can read those types from a file. Its initial value is:

```
((AC \READACFONTFILE)
 (STRIKE \READSTRIKEFONTFILE))
```

The functions take (STREAM FAMILY SIZE FACE) as arguments and return a CHARSETINFO datum. You will (probably) need the Xerox (internal) documentation about fonts and character sets (not supplied with the standard documentation) to define a new font file reading function.

The AC and STRIKE font types are handled specially to be compatible with the existing font code, so files with extension DISPLAYFONT still work and FONTFILEFORMAT moves the file pointer to the appropriately for those two types. For all other (new) types, the type is determined solely from the file extension and FONTFILEFORMAT has no side effects.

When defining a new display font types, you will need to add the new extension to the system list DISPLAYFONTEXTENSIONS.

---



---

## REGION

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

REGION facilitates having multiple complex cursor behaviors in a single window without having the CURSORMOVEDFNs, CURSORINFNs, CURSOROUTFNs, and BUTTONEVENTFNs of the behaviors know about each other. In its simplest form it can be used to implement active regions.

To use, set the various window functions of the window to the REGION window functions and put a list of REGIONEVENT records on the REGIONEVENTLST property of the window. When the cursor moves over the window, REGION checks which region it is in, calls the CURSOROUTFN of the previous region and the CURSORINFN of the new region. If regions overlap, then the appropriate functions will be called on all regions affected by the mouse event.

The REGIONEVENTLST property of the window should contain a list of REGIONEVENT records which have the fields:

EVENTREGION	A REGION record which is the region of the window over which the region specific functions will be invoked.
REGIONBUTTONFN	<i>(WINDOW POSITION REGION REGIONEVENT)</i>
REGIONMOVEDFN	<i>(WINDOW POSITION REGION REGIONEVENT)</i>
REGIONINFN	<i>(WINDOW REGION REGIONEVENT)</i>
REGIONOUTFN	<i>(WINDOW REGION REGIONEVENT)</i>
REGIONREPAINTFN	<i>(WINDOW REGION REGIONEVENT)</i>
ACTIVEREGION	Boolean indicating if the region is active or not.
REGIONFLAGS	User defined identification flags.
REGIONUSERDATA	User defined field.

All of the fields in the REGIONEVENT record are optional. If a REGIONEVENT record has a NIL EVENTREGION, then it is considered the default REGIONEVENT and will be invoked whenever a mouse event occurs outside of any other region.

The REGION window functions are:

```
(WINDOWPROP WINDOW 'CURSORINFN (FUNCTION REGIONINFN))
(WINDOWPROP WINDOW 'CURSOROUTFN (FUNCTION REGIONOUTFN))
(WINDOWPROP WINDOW 'REPAINTFN (FUNCTION REGIONREPAINTFN))
(WINDOWPROP WINDOW 'CURSORMOVEDFN (FUNCTION REGIONMOVEDFN))
(WINDOWPROP WINDOW 'BUTTONEVENTFN (FUNCTION REGIONEVENTFN))
```

The above window properties can be set using the function:

(REGION.INIT *WINDOW* [*REGIONEVENTLST* *SAVE?*]) [Function]

The *REGIONEVENTLST* is a list of *REGIONEVENT* records to put on the window. If *SAVE?* is non-NIL, the *CURSORINFN*, *CURSOROUTFN*, etc. of the window are put into a default region event record (one with an *EVENTREGION* = NIL) and added to the *REGIONEVENTLST*. The macro:

(ADDREGIONEVENT *REGIONEVENT* *WINDOW*) [Macro]

can be used to add a *REGIONEVENT* record onto the current *REGIONEVENTLST* of *WINDOW*.

The *REGIONFLAGS* field of the *REGIONEVENT* record consists of whatever atoms the user wishes to identify regions with. These allow the user to issue commands such as "turn off all regions marked *GRAPH*", "activate all the *MENU* regions", etc.

(ACTIVATEREGIONS *FLAGS* *WINDOW*) [Function]

(DEACTIVATEREGIONS *FLAGS* *WINDOW*) [Function]

Activate and deactivate all the *REGIONEVENT* records on *WINDOW* whose *REGIONFLAGS* have a flag in common with *FLAGS*. If *FLAGS* is T, activate or deactivates all *REGIONEVENT* records.

DISABLEFLG [Variable]

If set to T, disables all of the region functions for all windows using the *REGION* module. Alternatively, setting *DISABLEFLG* to a window, or list of windows, disables all the windows using the *REGION* package except for those windows. This allows selectively turning off cursor actions on parts of the screen.



---



---

**REMOTEPSW**


---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: COURIERSERVE, COURIERDEFS

REMOTEPSW defines a *remote* process status window facility that runs on top of Courier. The remote process status window is identical to the local one except that it contains **UPDATE** (to get the current process status) instead of **BREAK**, and **INFO** is not implemented. Both the client and server code are contained in the module which must be loaded on both hosts. The Courier server must be running on the host you wish to monitor.

The only user function is:

(REMOTE.PROCESS.STATUS.WINDOW *HOST*)

[Function]

which opens a remote process status window onto *HOST*, where *HOST* is any NS host specification that COURIER.OPEN accepts.

```

221 # 0.52612,100127 # 0
COURIER221#0.52612.100127#137624
COURIER.LISTENER+124745
SPP#221#0.52612.100127#5
COURIER.LISTENER
LAFITEMAILWATCH
ERIS#LEAF
\3MBWATCHER
MOUSE
Tedit#2
Tedit
EXEC
\NSGATELISTENER
\PUPGATELISTENER
\TIMER.PROCESS
BACKGROUND
BT      WHO?   KILL
BTV     KBD←  RESTART
BTV*    INFO   WAKE
BTV!    UPDATE SUSPEND

```

---



---

**RPC**  
**SUN REMOTE PROCEDURE**  
**CALLS**

---



---

By: JFinger

Supported by Atty Mullins (Mullins.pa@Xerox.com) and Bill Van Melle (vanMelle.pa@Xerox.com).

This document last edited on August 1, 1988.

### INTRODUCTION

This module implements SUN remote procedure calls as specified in the Remote Procedure Call Protocol Specification. The syntax is oriented toward Lisp users, differing greatly from Sun's C-like syntax.

#### RPC2 Package

All functions and variables mentioned in this document are defined as external variables in the package RPC2, unless otherwise stated.

#### REMOTE PROCEDURE DEFINITION

Remote programs are defined via calls to `define-remote-program`.

<code>define-remote-program</code>	<code>name number version protocol &amp;key :constants :types :inherits :procedures</code>	[Function]
	Defines parameters and result types of the procedures of remote program (number, version, protocol) . If successful, returns name, otherwise nil.	
name	a string or symbol that may be used by other procedures (for example, <code>remote-procedure-call</code> ) to uniquely specify this remote program.	
number	is the program number of this program on the remote machine. As specified in Sun's Remote Procedure Call Programming Guide, programs 0 - #x1ffffff are defined by Sun, #x20000000 - #x3ffffff are reserved for users, and #x40000000 - #x5ffffff are designated as transient.	
version	a number, is the desired version of remote program.	
protocol	an atom, UDP or TCP. ( At the moment TCP is not supported under Medley 1.0-5).	
constants	a list of pairs ( <code>&lt;constant-name&gt;</code> <code>&lt;constant-def&gt;</code> ), where <code>&lt;constant-name&gt;</code> is a symbol and a <code>&lt;constant-def&gt;</code> is an XDR constant (See XDR Constant Definitions below) .	
inherits	a list of name 's of other remote programs from which types and constants are inherited. Inherited types and constants are resolved by searching this list in order.	
types	a list of pairs ( <code>&lt;type-name&gt;</code> <code>&lt;typedef&gt;</code> ) , where a <code>&lt;type-name&gt;</code> is a symbol and a <code>&lt;typedef&gt;</code> is an XDR type definition (defined below).	

procedures a list of 4-tuples of the form (<procname> <procnumber> <arg-types> <result-types>), where <procname> is a symbol or string naming the procedure, <procnumber> is the procedure number on the remote machine, <arg-types> is a (possibly empty) list of XDR type definitions (see below) of the arguments to be sent to the remote procedure, and <result-types> is a (possibly empty) list of XDR type definitions of data to be returned from this remote procedure.

### XDR (EXTERNAL DATA REPRESENTATION) TYPE DEFINITIONS

Because the client and server machines may represent data in different ways, a data representation common to both machines is necessary. Remote procedure calls pass data between machines in 'External Data Representation' (XDR). The XDR language implemented here is oriented toward Lisp in its syntax and is not identical to the language spelled out in the Sun XDR Protocol Specification.

XDR data types may be defined in the :types keyword argument for later reference in the :types or :procedures of this or later remote programs. When a remote program is defined (usually at load time), the needed reading and writing functions are compiled for each constructed type referenced. Note that all XDR calls are eventually resolved to a composition of Primitive and Constructed XDR Type Definitions (see below).

### SYNTAX

The keywords of the XDR language may be specified as symbols of the Keyword package.

All XDR Data Types Definitions (notated here as a <typedef>), used in Remote Procedure Calls are from the following language:

- 1) **Primitive Definition:** One of the types in \*xdr-primitive-types\*,
  - :integer
  - :boolean
  - :unsigned
  - :hyperinteger
  - :hyperunsigned
  - :string
  - :float (not yet implemented)
  - :double (not yet implemented).
  
- 2) **Constructed Definition:** One of the types in \*xdr-constructed-types\*,
  - (:enumeration (<symbol-1> <constant-1>) ... (<symbol-n> <constant-n>))
  - (:union <enumeration-type> <typedef-1> ... <typedef-n>)
  - (:fixed-array <typedef> <constant>)
  - (:counted-array <typedef>)
  - (:opaque <constant>)
  - (:struct <defstruct-type> (<field-name-1> <typedef-1>) ... (<field-name n> <typedef-n>))
  - (:sequence <typedef>)
  - (:list <typedef-1> ... <typedef-n>).
  
- 3) **Local Definition:** A symbol defined previously in the same remote program definition.

Example: `:types ((nrec :unsigned)...) says that type 'nrec' is really only of type ':unsigned'.`

4) **Qualified Definition:** A dotted pair of the form (`<RPC program name> . <type>`), where `<type>` is an XDR type local to `<RPC program name>`.

Example: `:types ((count (myprog . nrec))...) says that a 'count' is really whatever myprog defines a 'nrec' to be.`

5) **Inherited Definition:** A symbol defined in the `:types` argument of a remote program R such that R is on the list of remote programs passed as the `:inherits` argument to the current remote program definition. The first such type definition found is used, that is, the list of inherited programs is scanned from left to right.

### XDR CONSTANT DEFINITIONS

Constants in XDR are defined by the following grammar:

`<constant-def> ::= <integer> | <defined-constant>`

`<defined constant> ::=`

- `<locally defined constant>`  
; Defined in the Remote Program currently being defined.
- `| <inherited constant>`  
; Defined in a remote program inherited by the current Remote Program (searched from left to right).
- `| <qualified constant>`  
; A dotted pair (`<rp>` `<constant>`), where `<constant>` is defined in remote program `<rp>`.

### SEMANTICS

An XDR type can be defined by a bidirectional filter mapping a subset of Lisp onto a byte stream and vice-versa.

For the XDR primitive type's filter, a description is given of its argument on the Lisp and XDR sides.

<code>:integer</code>	Lisp: an integer in range -2,147,483,648 to 2,147,483,648 inclusive. XDR: a 4 byte two's complement integer, high order to low order.
<code>:unsigned</code>	Lisp: an integer in range 0 to 4,294,967,295 inclusive. XDR: a 4 byte non-negative integer, high order to low order.
<code>:boolean</code>	Lisp: NIL for false, non-NIL for true. (The Lisp symbol T is returned when decoding a 1 from the XDR side.) XDR: 0 for false, 1 for true.
<code>:hyperinteger</code>	Lisp: an integer in range -(263) to 263 -1 inclusive. XDR: a 8 byte two's complement integer, high order to low order.

:hyperunsigned	Lisp: an integer in range 0 to 264-1 inclusive. XDR: a 8 byte non-negative integer, high order to low order.
:string	Lisp: a string of any length. XDR: Suppose the string is of length n. The XDR representation is an :unsigned (the string length n), followed by the n bytes of the string, followed by enough 0 bytes to make a multiple of 4 bytes.
:string-pointer	(UDP only) Lisp: a dotted pair (addr . nbytes), where addr is a buffer's address and nbytes is the number of bytes in the buffer. (Should I add an offset argument?). This is a speed hack to avoid having to copy VMEMPAGEP's twice. XDR: An XDR :string, as above.
:float	Lisp: A floating point number. (NOT YET IMPLEMENTED). XDR: A 4 byte floating point number in IEEE format.
:double	Lisp: A floating point number. (NOT YET IMPLEMENTED). XDR: A double precision floating point number in IEEE format.
:void	Lisp: null XDR: no bytes.

For each constructed XDR type, the declaration syntax is given along with its corresponding mapping.

(:enumeration (<symbol> <integer>) ... (<symbol> <integer>))	Lisp: a symbol XDR: an XDR :integer. The Lisp symbol (Each symbol is the "discriminant" for that value of the enumeration) and the XDR integer will be from a corresponding pair in the declaration. It is an error to try to encode a symbol not in the declaration or to try to decode an XDR integer for which there is not a corresponding symbol in the declaration.
(:union <enumeration-type> (<symbol-1> <typedef-1>) ... (<symbol-n> <typedef-n>))	Lisp: A list of two elements, the first being a discriminant for the enumeration type, and the second the appropriate Lisp input/output for the typedef corresponding to that discriminant's type.. XDR: An :integer discriminant followed by the XDR input/output for the typedef corresponding to that discriminant's type.
(:fixed-array <typedef> <constant>)	Lisp: An array of length <constant>, each element of which is an object of type <typedefLisp>. Note that since the function elt is used in encoding, any Lisp sequence could be used in place of an array. XDR: A sequence of <constant> objects of type <typedefXDR>.
(:counted-array <typedef>)	Lisp: A list of two elements, the first of which is an integer (the number of objects to be encoded/decoded), and the second of which is an array of objects of type <typedefLisp>. XDR: An integer (the number of objects to be encoded/decoded) followed by that number of objects of type <typedefXDR>.
(:opaque <constant>)	Lisp: A string of length <constant>.

XDR: A sequence of <constant> bytes followed by enough null bytes to round <constant> up to a multiple of four.

(:struct <defstruct-type> (<field-name-1> <typedef-1>) ...(<field-name n> <typedef-n>))

Lisp: A struct of type <defstruct-type> such that each field mentioned in the this XDR declaration has a value. Note that a separate defstruct must be executed. The fields need not be named here in the same order as those in the defstruct, nor must all the fields named in the defstruct be used here.

XDR: A sequence of objects of types <typedef1 XDR>...<typedefn XDR>.

(:sequence <typedef>)

This is fashioned after Courier 's method for encoding/decoding linked lists. This type can often be used to get around clumsy recursive definitions involving :union's of enumeration type :boolean.

Lisp: A list of objects of type <typedefLisp>.

XDR: A sequence of objects, each preceded by an XDR :boolean encoding of true. The last object in the sequence is followed by the XDR :boolean encoding of false.

Note: (:sequence <typedef>) produces the same encoding (but not the same decoding) as (defstruct astructure this-element the-rest) along with the declaration

```
(:recursive (:union      :boolean
                    (T (:struct astructure
                        (this-element <typedef>)
                        astructure)))
              (the-rest
               (NIL :void))),
```

(:list <typedef-1> ... <typedef-n>)

Lisp: A list, the ith element of which is of type <typedefi Lisp>.

XDR: A sequence of objects, the ith of which is of type <typedefi XDR>.

(:skip <unsigned>)

(For decoding only)

Lisp: Nothing

XDR: Any n bytes of data, where <unsigned> = n.

Note: This is a kludge for not having to decode the fattr's that NFS returns with every single cotton-pickin' memory read.

#### EXAMPLE OF A REMOTE PROGRAM DEFINITION

The following call to define-remote-program defines the portmapper remote procedures described in Sun's Remote Procedure Call Specification. Note that there are two definitions of procedure 4 given. Since remote procedures may be invoked by name, it is reasonable for there to be more than one definition for how to decode and encode the arguments to a given routine. In

this case, both a recursive and non-recursive definition is given for the values returned from procedure 4. Note also that mapstruct and mapsequence must be defstruct'ed before this call to define-remote-procedure.

```
(define-remote-program 'portmapper 10000 2 'udp
  :types '( (mapstruct (:union :boolean
                        (nil :void)
                        (t (:struct mapstruct
                           (program :unsigned)
                           (vers :unsigned)
                           (prot :unsigned)
                           (port :unsigned)
                           (therest mapstruct))))))
          (mapsequence (:sequence (:struct mapsequence
                                   (program :unsigned)
                                   (vers :unsigned)
                                   (protocol :unsigned)
                                   (port :unsigned))))))
  :procedures '( (null 0 nil nil)
                 (lookup 3 (:unsigned :unsigned :unsigned :unsigned)
                          (:unsigned))
                 (gooddump 4 nil (mapsequence))
                 (dump 4 nil (mapstruct))
                 (indirect 5 (:unsigned :unsigned :unsigned
                              :string)
                          (:unsigned :string))))
```

## UNDEFINING REMOTE PROGRAMS

undefine-remote-program name number version [Function]

## MAKING REMOTE PROCEDURE CALLS

remote-procedure-call destination program procid arglist [Function]  
 &key destsocket version credentials protocol  
 dynamic-prognum dynamic-version  
 msec-until-timeout msec-between-tries noerrorflg

Performs a remote procedure call to program on destination. Returns a list of the returned values.

**destination** Designates the host to which the procedure call is made. If Destination is a number it is interpreted to be the ip:iphostaddress of the host; if a symbol or string, it is a name from which the net address of the host may be found.

**program** Designates the remote program to be called. If Program is a number, it is interpreted to be the remote program number. If a symbol, in which case it is assumed to be the name of the remote procedure (as defined in define-remote-procedure. If :version is non-nil, then program is treated as a number rather than as a name. If version is nil and program is a number, then the latest version of that program is used.

**procid** Designates the procedure number from program to be called. If Procid is a number it is interpreted to be the remote procedure number; if a symbol, it is the name given that procedure in define-remote-procedure.

arglist	A list of the arguments to be serialized into XDR representation and passed as the arguments of the remote procedure call.
:destsocket	Normally, the remote socket must be looked up in the local caches (See <i>*rpc-socket-cache*</i> and <i>*rpc-well-known-sockets*</i> ) or else found by making a call to the Portmapper on the remote machine. If <i>:destsocket</i> is non-nil, its value is used as the remote socket.
:version	If non-nil designated the desired version of program as well as causing program to be interpreted as a number rather than a name. See program above.
:credentials	An object of type authentication to be passed as the credentials of the remote procedure call. (See <i>create-unix-authentication</i> ).
:protocol	A symbol specifying the transport protocol. Currently only UDP is implemented. Defaults to UDP. The only reason for using this parameter is to specify (along with the program and version), which known remote program is to be used.
:dynamic-program	If you really can't live without it, <i>dynamic-program</i> is used as the remote program number in spite of treating the arglist and returned values exactly as in program. Don't ask why.
:dynamic-version	If you really can't live without it, <i>dynamic-version</i> is used as the remote program version in spite of treating the arglist and returned values exactly as specified in program (and possibly version). Don't ask why. Defaults to 1.
:msec-until-timeout	Total number of milliseconds of waiting for a reply packet before giving up on this remote procedure call. Defaults to value of <i>*rpc-msec-until-timeout*</i> .
:msec-between-tries	Number of milliseconds between outgoing UDP packets. Defaults to <i>*rpc-msec-between-tries*</i> .
:errorflg	If <i>:noerrors</i> , ignores remote procedure call errors. If <i>:returnerrors</i> , returns the error as an s-expression. Otherwise, signals a Lisp error. Default t.

#### LOW-LEVEL REMOTE PROCEDURE CALL FUNCTIONS

setup-rpc	destination program procid &optional destsocket version protocol dynamic-program dynamic-version	[Function]
	Returns four values <i>destaddr</i> , <i>socket</i> , <i>program</i> and <i>procedure</i> (Yes, this is real, live multiple value return requiring a <i>multiple-value-bind</i> or something similar.) for consumption by <i>perform-rpc</i> . The arguments to <i>setup-rpc</i> are identical in meaning to the identically named arguments to <i>remote-procedure-call</i> .	
open-rpc-stream	protocol destaddr destsocket	[Function]
	Returns an <i>rpcstream</i> for use by <i>perform-rpc</i> . <i>Destaddr</i> and <i>destsocket</i> are as returned by <i>setup-rpc</i> and <i>protocol</i> is identical to the <i>protocol</i> argument to <i>remote-procedure-call</i> .	



close-rpc-stream                      rpcstream protocol                      [Function]

Closes rpcstream, an rpc-stream of protocol protocol created by open-rpc-stream.

perform-rpc                      destaddr destsocket program procedure rpcstream                      [Function]

arglist credentials protocol &key errorflg leave-stream-open msec-until-timeout msec-between-tries

Performs a remote procedure call returning a list of the values retruned by the remote procedure.

#### LISTING REMOTE PROGRAMS CURRENTLY DEFINED

list-remote-programs                      [Function]

Returns a list of 4-tuples (name number version protocol) for each remote program currently defined.

#### CREATION OF CREDENTIALS

create-unix-authentication                      stamp machine-name uid gid gids                      [Function]

Returns a Unix-type authentication suitable for use as the credentials of a call to remote-procedure-call or perform-rpc.

stamp                      An arbitrary unsigned integer.

machine-name                      A string containing the name of the calling machine.

uid                      User id number on the remote machine.

gid                      Group id number on the machine.

gids                      A list or array of group id numbers (on the remote machine) that contain the caller as a member.

#### GLOBAL VARIABLES

\*xdr-primitive-types\*                      An a-list of keywords and the corresponding function that implements that XDR primitive type.

\*xdr-constructed-types\*                      An a-list of keywords and the corresponding function that generates code to implement that XDR constructed type.

\*msec-until-timeout\*                      Number of milliseconds before giving up on receiving a reply packet. Default 1000.

\*msec-between-tries\*                      Number of milliseconds to wait before resending UDP packet. Default 100.

\*rpc-ok-to-cache\*                      If non-nil, uses \*rpc-socket-cache\* as a cache of socket numbers found to date.

- \*rpc-well-known-sockets\*** A list of well-known sockets. Format is  
 ( <host address>  
 <remote program number>  
 <remote program version>  
 <protocol>  
 <socket> )
- \*rpc-socket-cache\*** A list of non-well-known sockets. Format is same as  
 \*rpc-well-known-sockets\*.
- \*debug\*** If non-nil prints out debugging information. If a number,  
 the higher the number, the more information is printed.  
 Default nil.

**RPC FILES**

- RPC** Sets up the RPC2 Package and loads other RPC files. Loads  
 Portmapper remote program definition and executes it.
- RPCLOWLEVEL** Super low-level UDP/TCP functions added to Eric Schoen's  
 TCPUDP code.
- RPCOS** Low-level interface to Sun OS networking code .
- RPCSTRUCT** Structure definitions used by the other files. These are in  
 a separate file because they take so long to compile.
- RPCCOMMON** Common lookup functions and stream i/o functions used  
 by the other files.
- RPCXDR** External Data Representation (XDR). Code Generation  
 for XDR constructed types and XDR primitive functions.
- RPCRPC** Remote program definition and remote procedure calls.
- RPCPORTMAPPER** Definition of portmapper in UDP and TCP.

**KNOWN DEFICIENCIES**

- Floating point XDR types are not implemented.
- The view-packet utility is not documented and needs to be smarter about authentications.
- Fall through cases of XDR types UNION and ENUMERATE should be added.
- TCP is not supported under Medley 1.0-S, this should be in the next release.

**COPYRIGHT INFORMATION**

Copyright (c) 1987, 1988 Leland Stanford Junior University and Envos Corporation.

Written by Jeff Finger under support of National Institutes of Health Grant NIH 5P41 RR00785  
 to the SUMEX-AIM Computing Resource at Stanford University.

Modified to work under Medley 1.0-S by Atty Mullins.

---

---

SCREENPAPER

---

---

By: Larry Masinter (Masinter.pa@Xerox.com)

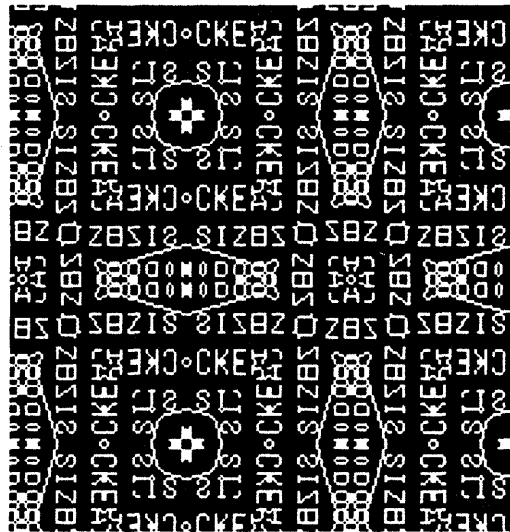
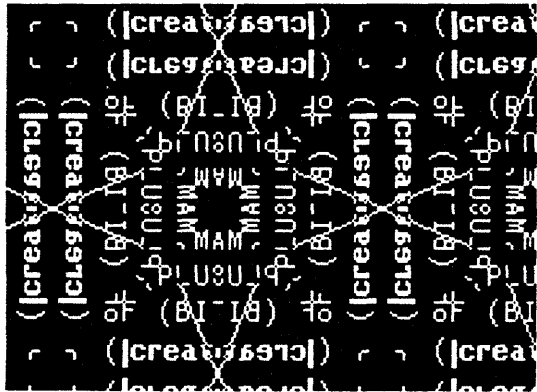
SCREENPAPER is an Idle hack ("Screen wallpaper"). Old fashioned wallpaper/wrapping paper from your screen.

Global parameters:

SCREENPAPERSIZE size of viewport, initially 64.

SCREENPERIOD how often to go into reflective move

SCREENREPEAT how long to stay in reflective mode (initially 0. e.g., disable reflective mode).



---

---

**SEdit-Menu-Always**

---

---

By: Matt Heffron (BEC.HEFFRON@ECLA.USC.EDU)

This package advises the SEdit Editor so that when an SEdit window is opened, the SEdit Attached Command Menu is automatically opened as well (depending on the setting of the global variable IL:SEditMenuAlwaysFlg). The value of IL:SEditMenuAlwaysFlg is initialized to T as an INITVAR when the file is loaded.

---



---

## SETDEFAULTPRINTER

---



---

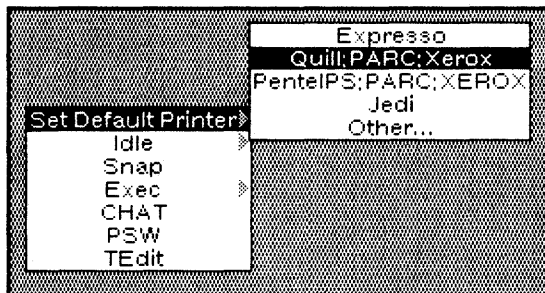
By: Nick Briggs (Briggs.pa@Xerox.com)

The SETDEFAULTPRINTER module provides a (cleaner) mechanism for moving printer names around on your DEFAULTPRINTINGHOST list. There are no user callable functions. Access to the features of the module are through the Background menu. This module uses the DEFAULTSUBITEMFN module which redefines the DEFAULTSUBITEMFN used in menus to accept an expanded form for menu subitems.

Set Default Printer

[Background Menu Entry]

Selecting the "Set Default Printer" item off the background menu will prompt you for a new default printer, which will be added at the beginning of the DEFAULTPRINTINGHOST list. If you roll-out into the subitems for Set Default Printer it will present a submenu with the entries on DEFAULTPRINTINGHOST, and an "Other..." item. Selecting one of the printer name entries will cause it to be moved to the front of DEFAULTPRINTINGHOST, selecting "Other..." will prompt for the name of a printer in the same manner as selecting the "Set Default Printer" top level item off the background menu. If any commentary information has been supplied (see below) holding the mouse over the printer name will display the information in the prompt window.



SDP.PRINTERINFO

[Variable]

The variable SDP.PRINTERINFO is an A-list which will be used to lookup commentary information about a printer to be included as the "help" in the menu subitems. The UPPERCASE name of the printer is used as a key. An example SDP.PRINTERINFO setting might be

```
((QUAKE . "Press, Rm 1532") (PENDELPS:PARC:XEROX . "Interpress, Rm 1532"))
```

LOCATION

[Property]

The code that looks up the commentary information about a printer will also check for a LOCATION property on the UPPERCASE atom which is the printername if no entry is found on SDP.PRINTERINFO. For example

```
(PUTPROP 'JEDI 'LOCATION "FullPress, Pod 5, 2nd floor")
```

Would describe the location of printer Jedi.

---



---

## SHOWTIME

---



---

By: Timothy Bigham (TBigham.henr@Xerox.com)  
 Medley mods by: Ron Fischer (Fischer.PA@Xerox.com)

Uses: BITMAPFNS, SCALEBITMAP, READBRUSH

This document last edited on May 13, 1988.

### INTRODUCTION

SHOWTIME provides a user interface to read, write, and edit bitmaps in several different formats. Among the supported formats is RES, used in VIEWPOINT Freehand Graphics. Other supported formats include Brush (Mesa Doodle format); and Lisp.

SHOWTIME has been written to readily accomodate new formats. Users may add new formats to Showtime by writing format-specific read and/or write functions and adding them to those Showtime knows about (described below).

Selecting SHOWTIME from the background will provide the user the opportunity to specify an area to use as the SHOWTIME window. After the user creates a SHOWTIME window, a left mouse button within the window will popup a menu of available options. There may only be one bitmap displayed in a SHOWTIME window at a time, but any number of SHOWTIME windows may be opened. Shrinking a SHOWTIME window will create an icon with the name of the bitmap that is displayed in the window.

### Functions, Variables, and Lisp Code Examples

SHOWTIME.FORMAT.FNS [Variable]

A global association list that maintains a list of all the formats Showtime knows about and the read and write functions to use with those formats. This variable should be updated by calling the function SHOWTIME.ADD.FORMAT to ensure successful integration of any new bitmap formats.

SHOWTIME.DEFAULT.FORMAT [Variable]

A variable that is initially set to 'LISP. This format uses the binary storage routines found in the lispusers module BITMAPFNS.

(SHOWTIME.ADD.FORMAT *FORMAT READFN SAVEFN*) [Function]

A function that should be called when the user wants Showtime to know about new bitmap formats. *FORMAT* may be any descriptive atom, such as RES or LISP. *READFN* and *SAVEFN* must be functions that have as the first two arguments FILENAME and BITMAP. In addition, the READFN must return a bitmap.

For example, the READFN code for LISP format is:

```
(LAMBDA (FILENAME)
```

```
(* this function must <1> read a bitmap from a file and <2> return the  
value of the bitmap)
```

```
(READBM (OPENFILE FILENAME (QUOTE INPUT))))
```

For example, the WRITEN code for LISP format is:

```
(LAMBDA (FILENAME BITMAP) (* TBigham "30-Dec-86 13:09")
```

```
(* this function must write a bitmap to a file)
```

```
(WRITEBM (OPENFILE FILENAME (QUOTE OUTPUT)) BITMAP))
```

### **ACKNOWLEDGEMENTS**

SHOWTIME was originally developed to provide a user interface in the exchange of bitmaps between the VIEWPOINT and INTERLISP-D environments. Tom Wall initiated the idea to exchange bitmaps between these environments and was instrumental in developing the code to write RES files. Gary Gocek originally wrote the code to read RES files; Mitch Garnaat wrote the code to write BRUSH files; reading brush files is supported by the lispusers module READBRUSH written by Larry Masinter. The SHOWTIME icon was designed and created by Mary Baecher-Cocca.

---

---

**SIMPLECHAT**

---

---

By: Larry Masinter (Masinter.PA@Xerox.COM)

Uses: TEDIT

This document last edited on Sept. 8, 1988.

### INTRODUCTION

Like CHAT except that it works in the current window/exec instead of spawning a new window. To exit from TTYCHAT there is an escape character, control-right-bracket (↑]). If you type ↑], you get prompted for a Chat command. This can be one of Binary, Text, or Close. Normally TTYCHAT translates incoming characters and converts EOL; setting Binary mode disables this. Close will close the connection.

### MODULE EXPLANATIONS

The CHATSERVER module advises CHAT to use TTYCHAT when the main "terminal" is not the display. This allows one to use the Lisp system as a "protocol translation gateway"; for example, on a Sun with CHATSERVER-NS loaded, you can Chat to the Sun using NS and then use UNIXCHAT to CHAT(SHELL).

(TTYCHAT *&optional host logoption*)

[Function]



---

---

## SOLID-MOVEW

---

---

By: Lennart Lövstrand  
(Lovstrand.EuroPARC@Xerox.COM)

This document last edited on May 13, 1988.

### INTRODUCTION

This module changes the behaviour of MOVEW when no destination is given to let the whole image of a window track the mouse instead of just its outline. To avoid flickering and to give an illusion of a smooth animation, all rendering operations are done off the screen, ending with a single bitblt to the frame buffer for each cycle. This can easily be done on small windows such as icons, but the more bits there are to be moved, the longer it takes to do the animation updates and the slower it becomes to solidly move windows. Therefore, the user can control when solid vs. outline moving is to be done by setting `*SOLID-MOVEW-FLAG*` to an appropriate value. By default, only windows containing less than 15,000 pixels will be moved solidly; all other windows are moved using the original MOVEW method.

SOLID-MOVEW interfaces nicely with both ICONW and ATTACHEDWINDOWS by being able to move images of arbitrary shape — not just pure rectangles. It also knows about GRID-ICONS and can be made to force the icons to snap to grid positions while being moved, thus producing a kind of jagged feeling. Finally, a shadow has been added emphasize the 2½-D property of window systems and to give a clear indication of when the window is in the process of being moved.

### PROGRAMMER'S INTERFACE

When loaded, the module replaces the system MOVEW function with its own version and moves the original code to ORIGINAL-MOVEW. The control and interaction is then comes through the following variables:

`*SOLID-MOVEW-FLAG*` [Variable]

This variable controls whether the new MOVEW should use solid or outline moving. It should have one of the following types of values:

- a NUMBERP Only use solid moving on windows that have a total size (width x height) less than or equal to the given number of pixels.
- a POSITIONP Only use solid moving on windows that have a width and height less than or equal to the two numbers.
- ICON Only move icons solidly. A window is considered to be an icon if it either has an ICONFOR or an ICONIMAGE property.
- T Move all windows solidly.
- NIL Move all windows using outlines.

The default value for `*SOLID-MOVEW-FLAG*` is 15000.

**\*SOLID-MOVEW-SHADOW\*** [Variable]  
**\*SOLID-MOVEW-SHADOW-SHADE\*** [Variable]

These two variables define whether or not a shadow should accompany the moving image. The shadow is always directed towards south-east and the first variable, **\*SOLID-MOVEW-SHADOW\***, determines its position by taking on any of the following types of values:

a NUMBERP	The x and y offsets of the shadow (same)
a POSITIONP	The x and y offsets of the shadow (different)
T	Use the default shadow offset — 3 pixels in both directions.
NIL	Don't show a shadow.

The second variable, **\*SOLID-MOVEW-SHADOW-SHADE\***, sets the darkness of the shadow, ie. the texture to be added to the background where the shadow is visible.

The default values for the two variables are T and 42405, a 50% gray shadow offset by 3 pixels.

**\*SOLID-MOVEW-GRIDDING\*** [Variable]

When used together with the **ICON-GRIDS** module, **SOLID-MOVEW** can be made to only move solid window images on grid positions, thus creating a kind of "jagged" feeling when interactively moving icons on the screen. If this is disabled, the icon will "snap" to the closest grid position only after the move has been completed.

The default value for **\*SOLID-MOVEW-GRIDDING\*** is NIL, thus disabling early gridding.

**\*SOLID-MOVEW-CASHING\*** [Variable]

**SOLID-MOVEW** uses separate bitmaps for rendering purposes so as to produce a smooth animated move and avoid unnecessary flickering on the screen. To speed up the initial phase of the move operation, the rendering bitmaps can be cached from one invocation to another. This will use up some bitmap space, but can be freed using (**GAINSPACE**) if need arises.

The default value for **\*SOLID-MOVEW-CASHING\*** is T, thus enabling cached rendering bitmaps.

(**SOLID-MOVEW POSorX Y**) [Function]

Because only those windows meeting the requirements of **\*SOLID-MOVEW-FLAG\*** will be moved solidly, the user has the option of calling **SOLID-MOVEW**. It takes the same arguments as **MOVEW**, but if either **POSorX** or **Y** is specified, control is again turned over to the old **MOVEW**.

If you get tired of all this, you can undo the behaviour of **SOLID-MOVEW** by typing the following form into an Interlisp Exec:

```
(MOVD 'ORIGINAL-MOVEW 'MOVEW)
```

## BUGS

No provision has been made to make **SOLID-MOVEW** work with color.

If the window is closed as a side effect of the its **MOVEFN** or **AFTERMOVEFN**, it will be reopened before **SOLID-MOVEW** returns

---

---

**SOLITAIRE**

---

---

By: Beau Sheil. Upgraded for Medley by Larry Masinter (Masinter.PA@Xerox.COM)

The SOLITAIRE package is a simple graphics demonstration program that plays and animates the solitaire card game (known as "Patience" in English speaking countries). Solitaire is a game for one, so there is no way to play "against" the machine. SOLITAIRE is most effective as a background activity when the machine is doing nothing else, so it is frequently used as an IDLE hack.

**TO USE****To play once**

(SOLITAIRE SOLOW REPLAY)

[Function]

Plays one hand of solitaire, which it will animate in the window SOLOW (which should be at least 700 by 700, although the program will do its best to adapt). If REPLAY is T, SOLITAIRE will use the deck from the previous shuffle, else it will deal a new hand.

**To play repeatedly**

(SOLO SOLOW)

[Function]

Calls (SOLITAIRE SOLOW) repeatedly.

**The results**

SOLO keeps a record of the frequency of each of its results in the array SOLORESULTS [0..52] which it plots at the end of each hand.

**As an IDLE hack**

Loading SOLITAIRE automatically adds SOLITAIRE as an option to the IDLE menu. If chosen, it will be given the "whole screen" covering window of IDLE and will use a black background, rather than its usual shaded one, to preserve the screen phosphor. Otherwise, its operation is completely normal.

---



---

STARBG

---



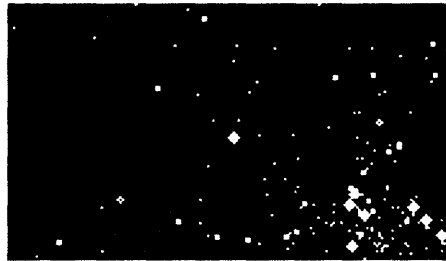
---

By: Gregg Foster (Foster.PA@Xerox.COM)

Upgraded for Medley by Larry Masinter (Masinter.PA@Xerox.COM)

STARBG creates a random star field for your screen background and a little flying saucer to follow your cursor when it's in space (so it doesn't get lost). It also supplies an alternate IDLE function, Cosmos.

The star field will look something like this:



The saucer will look like this:



**USAGE**

(STARBG) [Function]

STARBG fills a screensized bitmap with random stars, turns the saucer on, and calls CHANGEBACKGROUND. If you don't like the star pattern you get, try it again.

(Cosmos window) [Function]

Cosmos is puts an evolving universe in a window. It's intended as an IDLE function, but will entertain you for hours in any decently sized window.

(SaucerOn) [Function]

SaucerOn turns the saucer on by changing the CURSORBACKGROUND\*FNs.

(SaucerOff) [Function]

SaucerOff turns the saucer off and sets the BACKGROUNDCURSOR\*FNs to NIL.

**CUSTOMIZATION**

There are lots of user-settable parameters, all of which have reasonable defaults. Here are some of the interesting ones:

STARBGParameters	[Variable]
is a list of settable parameters. Most are dotted pairs specifying ranges (e.g. stars3 defaults to (6 . 70) meaning that STARBG will make 6 to 70 type-3 stars). The others are bitmaps.	
BM1, ..., BM5	[Variables]
The star bitmaps used to BLT the stars. BM1 must be a single bit.	
SBM	[Variable]
The starry screen bitmap. This is reused in subsequent calls to STARBG.	
stars1, ..., stars5	[Variables]
Ranges for the 5 kinds of stars.	
constellations	[Variable]
Range for number of constellations. A constellation is a group of bright stars.	
clusters	[Variable]
Range for number of clusters. Clusters are tightly globular.	
superClusters	[Variable]
Range for number of superClusters. SuperClusters are clusters of clusters.	
eventPause	[Variable]
Number of milliseconds to block between events. Larger numbers have the effect of slowing down the rate of evolution..	
changeStars	[Variable]
Will use the IDLE-ing star field as your new background.	

---

---

**STEP-COMMAND-MENU**

---

---

By: Matt Heffron (BEC.HEFFRON@ECLA.USC.EDU)

This package changes the function CL::STEP-COMMAND (used by CL:STEP) to call a new function (instead of IL:ASKUSER) to get its commands from a menu attached to the stepping window (depending on the setting of the CL:SPECIAL variable IL:\*STEP-COMMAND-MENU\*). The value of IL:\*STEP-COMMAND-MENU\* is initialized to T as an INITVAR when the file is loaded. The variable USER:\*STEP-COMMAND-INVERT-MENU-SHADE\* is the shade used to *grey-out* the attached menu when the stepping is not awaiting a command. The menu is attached to the Right edge (at the Bottom) of the stepping window. (If there isn't enough room on the Right, it will be attached to the Left edge.) The menu is detached and closed when the stepping level which first attached it is exited.

STORAGE

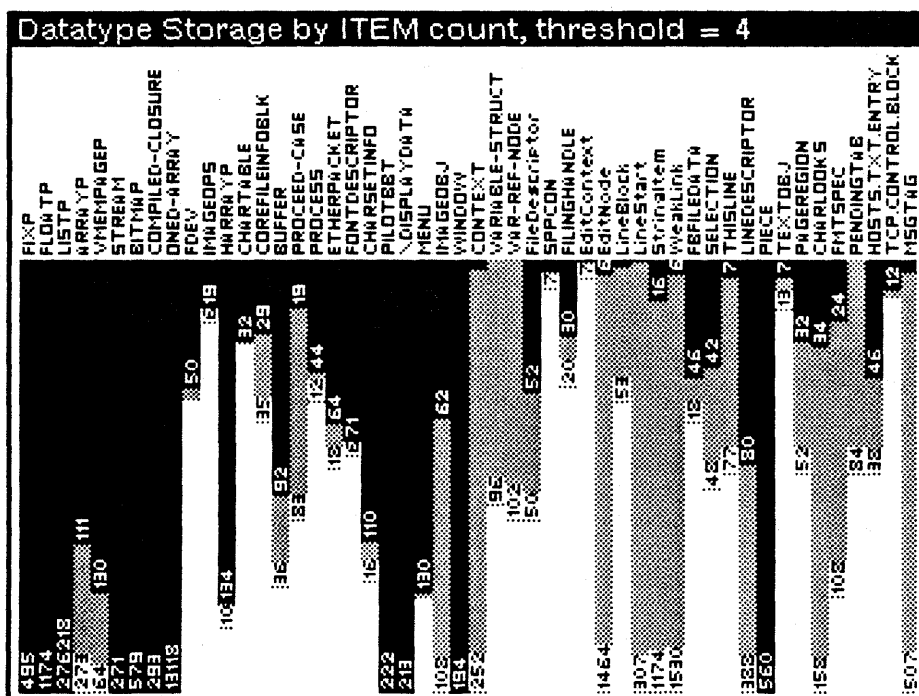
By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

STORAGE implements a bar-graph version of the Lisp STORAGE function, providing a visual summary of the amount of storage allocated to each data type.

(SHOWSTORAGE [PAGETHRESHOLD MODE ROTATION])

[Function]

Displays the storage allocation of Lisp data types in bar graph format:



All the arguments are optional. *PAGETHRESHOLD* is the same as for the STORAGE function and defaults to 1. *MODE* determines what to display and can be one of the following:

- ITEM** The number of items of each type that have been allocated (the default mode).
- PAGE** The number of pages allocated for each type.
- BOX** The number of times each type has been allocated (see *BOXCOUNT* in the IRM).

The mode can be changed when the window is open by clicking with the *middle* mouse button. Clicking in the window with the *left* mouse button will update the window. When the window is redisplayed (using the standard window menu or *REDISPLAYW*) it will add new data types that have been defined since the window was last redisplayed.

For the **ITEM** and **PAGE** modes, the black part of the bar represents the number of items or pages currently *in use*. The gray part of the bar represents the number of *free* items or pages. The total length of the bar represents the *total* number of items or pages.

The *ROTATION* argument can be one of **NIL** (use the rotation of the **SHOWSTORAGEFONT**), **0** (labels from bottom to top on the right, bars grow to the left) or **90** (labels from left to right and bars grow down).

The display is controlled by the following global variables:

**SHOWSTORAGEWINDOWSIZE** [Variable]

The width or height (depending on the rotation) of the window, initially 275 (pixels). The bars truncate at the edge of the window; the window can be reshaped to put the longer bars in perspective.

**SHOWSTORAGEIGNORE** [Variable]

A list of data types to ignore. The information for the data types initially on this list is incorrect and/or their inclusion breaks the program.

**SHOWSTORAGEDEFAULTTHRESHOLD** [Variable]

The default threshold used when *PAGETHRESHOLD* is **NIL**, initially 1 (page).

**SHOWSTORAGEPRIN2FLG** [Variable]

Flag that causes **PRIN2** to be used instead of **PRIN1** when printing data type names (**PRIN2** will include package names), initially **NIL**.

**SHOWSTORAGEFONT** [Variable]

The window font, initially one of Helvetica 5 through 10, i.e. the smallest that can be found when the file is loaded. The default font has a rotation of *90* degrees.



---



---

## SYSTATS

---



---

By: Johannes A. G. M. Koomen  
(Koomen.wbst@Xerox or Koomen@CS.Rochester)

This document last edited on: October 28, 1987

### SUMMARY

SYSTATS provides a functional interface to system statistics such as PageFaults, DiskIOTime, etc. Statistics are maintained in objects of type SYSTATS. Functions are provided to fetch values from these objects, and to update the objects to reflect the current system state or to compute differences. This facility provides a Lyric alternative to the (undocumented) MISCSTATS functions in Koto.

### DESCRIPTION

SYSTATSPROPS [Variable]

A list of statistics maintained by SYSTATS. Changing it does not alter SYSTATS behavior.

(SYSTATSPROP *prop fromstats*) [Function]

If *fromstats* is NIL, the internal SYSTATS object is updated and used. Returns the value of the statistic named by *prop*, which must be a member of the variable SYSTATSPROPS.

**Caveat:** The value returned is a FIXP which is an element of the *fromstats* object and which, for the sake of performance, is reused during a SYSTATSREAD on the *fromstats* object. Note that there is an implicit SYSTATSREAD on the internal SYSTATS object if *fromstats* is NIL.

(SYSTATSREAD *intostats fromstats*) [Function]

If *intostats* is NIL, it is set to a newly created SYSTATS object. If *fromstats* is NIL, the internal SYSTATS object is updated and used. Copies system statistics from *fromstats* into *intostats*. Returns *intostats*.

(SYSTATSDIFF *oldstats newstats difstats*) [Function]

If *oldstats* is NIL, the internal SYSTATS object is updated and used in its place. If *newstats* is NIL, the internal SYSTATS object is updated and used in its place. If *difstats* is NIL, it is set to a newly created SYSTATS object. Computes the statistics differences between *oldstats* and *newstats*, and places the results in *difstats*. Returns *difstats*.

(CLOCKTICKS *interval timerunits*) [Function]

Returns the (machine dependent!) number of internal clock ticks over the interval. For instance, on the D'Lion, (CLOCKTICKS 2.5 'MINUTES) = 5211900.

---



---

## TALK

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)


Uses: Various editor and network protocol modules.

TALK allows users to hold conversations between machines across the Ethernet. TALK uses various *services* (TTY, TEdit and Sketch) and network *protocols* (NS and IP).

```

TALK (TEdit)
Howdy,
did you finally get these to load?

ELEMENTS --- BOX
           --- CIRCLE
           --- POLYGON
           --- SPLINE
           --- TEXT

No I haven't, where are they?
(NS) Talk from Christopher D. Lane;
Greetings,
not yet. Did you see the new bitmaps:

They're in the IMAGES directory.

Disconnect | RingBells | Message

```

### TALK FILES

Talk's services and protocols are now in separate files which may be loaded independently:

TALK            The main Talk module.

#### Services

TTYTALK        Simple text conversation between machines running Lisp, XDE and Viewpoint.  
TEDITTALK      Uses TEDIT; allows the full capabilities of the TEdit editor in a conversation.  
SKETCHTALK    Uses SKETCH; allows a conversation using the Sketch graphics editor.

#### Protocols

NSTALK        Uses COURIERSERVE (and optionally NSTALKGAP); allows XNS protocols.  
NSTALKGAP     Used by NSTALK if the GAP Courier program has not been defined (by NSCHAT).  
IPTALK        Uses TCP and TCPUDP; allows conversations using IP protocols.

Any Talk service can be used with any Talk protocol. The preferred order of loading is:

(FILESLOAD TALK TEDITTALK TTYTALK SKETCHTALK NSTALK IPTALK)

dropping out those services/protocols you do not use. Order of loading determines which services/protocols are tried first; the files may be loaded in any order to force different priorities.

## USING TALK

(TALK [USER.OR.HOSTNAME SERVICE PROTOCOL]) [Function]

Starts a TALK session; *USER.OR.HOSTNAME*, *SERVICE* and *PROTOCOL* are optional. If not supplied, *USER.OR.HOSTNAME* is prompted for (either by menu or typein or both). If *SERVICE* and *PROTOCOL* are not supplied (the usual case), TALK will figure out which to use based on what is available on the local and remote machines. The supported services and protocols are described below. The service and protocol used are indicated in the title bars of the TALK window.

TALK returns a process handle if the connection is successfully opened; it returns NIL if the user aborts out of the host/user menu and it returns an error message (as a string or list instead of breaking) if it cannot contact the remote host (for whatever reason). TALK can also be invoked from the background menu.

## TALK MENU

The menu at the bottom of the TALK window (which is only active while the connection is open) contains the following items:

- Disconnect** Closes the TALK connection. This is equivalent to closing the TALK window, but leaves the window open in case you want to save and/or hardcopy part or all of the session.
- RingBells** Rings the bell on the remote workstation (if possible) and flashes the TALK window on the local one to indicate it has done so. This is useful if you have asked a person to hold and want to let them know you have returned.
- Message** Prompts for and inserts a *canned* message into the TALK stream. Useful if the phone rings and you want to ask the other person to hold with a minimum of time/effort. Messages can be added to the list, see the TALK.USER.MESSAGES variable below. The TALK window must have the keyboard in order to use this item.

## ERROR MESSAGES

The TALK function will return one of the following error messages when it fails to start a session:

- Host not found!** It could not find host address for the host or user name specified.
- Can't connect to host!** The remote workstation does not have the appropriate server loaded and/or running or does not have TALK loaded.
- No answer from TALK service!** A connection was made, but no one responded (intentionally or otherwise). A darkened TALK icon is left on the remote screen to log the connection attempt (unless TALK.GAG is non-NIL).
- Unknown service type!** An unknown type was given as the *SERVICE* argument.
- No services available!** The *SERVICE* argument was not supplied and it cannot find one.
- Unknown protocol!** An unknown protocol was given as the *PROTOCOL* argument.

**No protocols available!**      The *PROTOCOL* argument was not supplied and it cannot find one.  
Service and protocol errors may indicate additional files need to be loaded.

### RECEIVING TALK

When your machine is contacted by another via TALK, the following icon will appear on your screen, ringing bells, flashing and showing the time, mode (service and protocol) and (when possible) the caller's identity:

TALK FROM:
Lane
TIME:
1-Aug 08:01
MODE:
TEdit(NS)

If you button the icon with the left or middle buttons, a TALK session will begin. If you either close the icon or do not button it (it will go *dark* (invert) in about 15 seconds if not buttoned) the TALK connection will be refused. TALK connections are automatically refused if the TALK.GAG flag is non-NIL (settable using the subitem(s) of the TALK item in the background menu). If the machine is in IDLE, TALK will wait twice the normal time out for the user to respond.

If you button a darkened (unanswered) TALK icon, it will try to reconnect you to the caller (after a mouse confirm). If a TALK connection comes in from someone who has already left an unanswered TALK icon on your screen, the icon will be reused.

### TALK SERVICES

#### TEdit

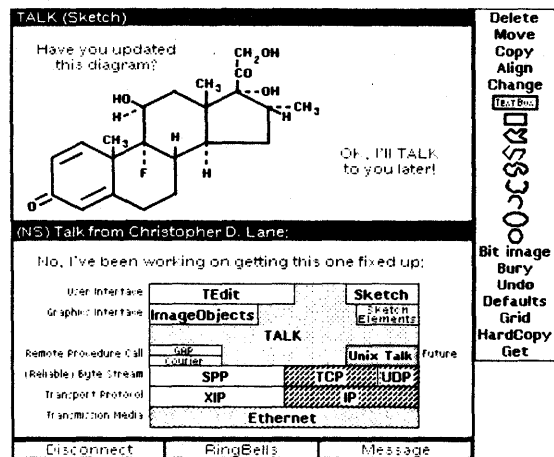
This service allows you to use the full capabilities of TEdit in your conversation, including: correcting mistakes anywhere in the document, changing character and paragraph looks, inserting ImageObjects, etc. Along with keyboard input, mouse selections and the caret are also visible to the remote user. The **GET** and **INCLUDE** commands in the TEdit command menu will load files into both the local and remote TEdit windows, so make sure the files are accessible to both. Similarly for fonts, if your workstation has to load a font from a server, the remote workstation must also have access to the font. Since the remote workstation may also need to load the font, you may experience communication delays. The TEdit service supports NS character codes and most of the 1108 and 1186 function keys.

#### TTY

This service is similar to the TEdit service except that the only supported feature is *backspace* (but not across lines). TTY is the only service that can talk with the Talk.bcd program in XDE or the TALK application (VPTalk.bcd) in Viewpoint. You *do not* need to know what type of workstation you are contacting when using any of the TALK programs.

## Sketch

The Sketch service is built on the Sketch graphics editor:



## TALK PROTOCOLS

### NS

When NSTALK is loaded, TALK will accept as a host name anything that COURIER.OPEN will accept including an NS address or the name of a workstation registered in a Clearinghouse. Additionally, user names can be used if the address of the user's workstation is registered under the user's name in the Clearinghouse. The following function can be used to register a user and workstation correspondence in the Clearinghouse:

(CH.USER.WORKSTATION *USER WORKSTATION*) [Function]

Sets (or changes) the AddressList Clearinghouse property of *USER* (which must already be a name or alias in the Clearinghouse) to be the address of *WORKSTATION* (an NS address or name). If *WORKSTATION* is NIL, the function removes the AddressList property from *USER*. To use this function, you must be logged in (via (LOGIN)) as a System Administrator for *USER*'s domain.

One way to register users would be to go to the individual's workstation, login as the System Administrator and evaluate: (CH.USER.WORKSTATION 'UserName \MY.NSADDRESS)  
Note that you cannot use the USERNAME function in this example since the (LOGIN) will change it.

NSTALK *does not* require or use NSCHAT, but they do share the Courier program GAP. If both NSTALK and the NS CHATSERVER modules are to be loaded, the CHATSERVER should be loaded *first* if possible. NSTALK is designed to allow other types of NSCHAT/GAP servers. The GAP server function determines which function to call using the service type requested (TTY = 5, TEdit = 6, Sketch = 7) and the entries on the association list GAP.SERVICETYPES which has entries of the form (ServiceNumber ServiceName ServerFunction). It is possible to have both NSTALK running and an EXEC server by adding appropriate entries to GAP.SERVICETYPES. If a GAP server already exists when NSTALK is loaded, it is made the default for all unrecognized service types.

Although NSTALK loads the COURIERSERVE LispUsers module you do not have to have a Courier server running to initiate an NS TALK connection, but you *must* have one running in order to receive an NS TALK connection.

**IP (Interim)**

When IPTALK is loaded, TALK will accept as a host name anything that DODIP.HOSTP will accept, including symbolic and numeric IP addresses. User names can be used by adding them as synonyms for local workstation hosts in the HOSTS.TXT file.

The current TALK IP interface is *only temporary* and will eventually be replaced by one which is compatible (for TTY service) with the TALK program which runs under BSD Unix; at that time, the allowable username format may be expanded to handle user@host. The current IP interface will probably not be compatible with the eventual, Unix-compatible one.

**TALK VARIABLES**

The following variables can be used to affect TALK's default behavior:

TALK.DEFAULT.REGION = (0 0 500 500) [Variable]

The LEFT and BOTTOM of this region determine where the (initial) TALK icon appears on the screen; the HEIGHT and WIDTH are the combined dimensions of the TALK windows (each uses half the HEIGHT). If this variable is set to NIL, then the icons start at (0 . 0) and the TALK window region is prompted for as needed.

TALK.USER.MESSAGES [Variable]

A list of menu items to put up when the MESSAGES item on the TALK menu is selected. Items on the list should return strings to be put into the TALK stream. If there is an entry of the form (GREETING "message") on this list, it will be printed automatically when a connection is opened.

TALK.GAG = NIL [Variable]

If non-NIL, causes the TALK server to automatically reject any TALK connections.

TALK.ANSWER.WAIT = 15 [Variable]

The number of seconds the TALK icon remains up before closing and aborting the connection.

TALK.HOSTNAMES = NIL [Variable]

A list structure containing hosts TALK has connected to along with the address used.

TALK.SERVICETYPES [Variable]

This list determines which services are tried and in what order. You only need to modify this if you wish to force an order other than the one determined by the order files were loaded or you wish to add or drop a service.

TALK.PROTOCOLTYPES [Variable]

This list determines which protocols are tried and in what order. You only need to modify this if you wish to force an order other than the one determined by the order files were loaded or you wish to add or drop a protocol.

**KNOWN PROBLEMS****Talk**

- Since TALK uses the Dove/Dandelion sound generator to help announce a connection, on other machines it is difficult for the user to detect connections being made during IDLE.

### TTY Talk

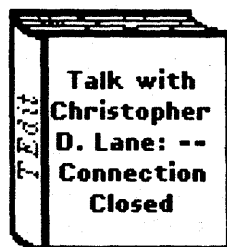
- The TTY service cannot backspace beyond the left margin (unlike other implementations).

### TEdit Talk

- Sometimes the local and remote TEdit windows will get out of sync as to what the current *looks* are; usually this is not serious.
- Page layout commands have not been implemented for the remote TEdit window; there are probably other commands that do not work either.
- ImageObject specific manipulations to ImageObjects already in the window do not get transmitted to the remote Tedit window.
- Inserting (other than keyboard input) into a pending delete does not echo correctly on the remote Tedit window.
- A large ImageObject inserted into the TALK window may not be seen by the remote user until some text is typed to force the remote window to scroll. The remote user may not see the ImageObject at all if it is larger than his window. These are both true of any TEdit window.
- User scrolling of the TEdit window will not cause scrolling of the remote TEdit window. System scrolling of the window (due to insertions and deletions) will be tracked in the remote window.

### Sketch Talk

- When the TALK window is opened, some sketch menus will be created and then replaced. This is due to Sketch not allowing a user to specify both an existing window and an initial menu.
- When text (or a text box) is entered, only the initial character is seen in the remote window until the text is completed and the user buttons some other point in the window.
- Arrow heads do not show up at all on the remote sketch window.
- Put of a SKETCHTALK sketch gets into an infinite loop so temporarily you must copy the sketch items to another sketch if you wish to save them on a file.
- If you *sweep* a control point on a box past the other one (like sweeping one corner of a region past the other in RESHAPE), the remote box will not move identically.
- Since there are no functions to programmatically manipulate grouped elements the **Group** and **UnGroup** items have been disabled in the Sketch Talk window.
- For a small number of changes (text fonts, text box brushes and closed wire dashing), the entire remote sketch window is redisplayed to make the change visible.
- Setting the SKETCHINCOLOR flag to a non-NIL value will cause some operations to break.



---



---

## TCPTIME

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: TCP, TCPUDP

TCPTIME implements time client and server routines under TCP/IP and UDP/IP based on RFC868. The following are the user functions; the *PROTOCOL* argument refers to one of **TCP** or **UDP** and defaults to the value of `RFC868.DEFAULT.PROTOCOL`, initially **TCP**. All arguments are optional:

`(RFC868.SETTIME [RETFLG PROTOCOL])` [Function]

Obtains the time from the network, similar to the `\PUP.SETTIME` and `\NS.SETTIME` functions. If *RETFLG* is non-NIL, the time is returned as an integer (as specified in RFC868), otherwise `SETTIME` is called and the new time is printed in the prompt window. Either `TCP.TIME.HOSTS` and/or `UDP.TIME.HOSTS` (see below) must be set before calling this function.

`(RFC868.START.SERVER [PROTOCOL ASCIIFLG])` [Function]

Starts a network time server process for the specified (or default) *PROTOCOL* if one is not already running. The *ASCIIFLG* is discussed below.

`(RFC868.STOP.SERVER [PROTOCOL])` [Function]

Deletes the network time server process for the specified (or default) *PROTOCOL* if one is running.

The following variables are used by the functions above:

`RFC868.TIME.PORT = 37` [Variable]

Used to set the initial value of the protocol specific port variables when the file is loaded. Once the file is loaded, changing this variable has no effect, so it must be reset (if necessary) before loading the file, otherwise the protocol specific port variables should be reset directly. See `TCP.TIME.PORT` and `UDP.TIME.PORT` below.

`RFC868.DEFAULT.PROTOCOL = TCP` [Variable]

The default protocol to use when one is not specified.

### **BINARY & ASCII TIME FORMAT**

Some network software implements the RFC868 standard by returning the printed (ASCII) representation of the time, rather than the binary representation as specified in the RFC. To work around this, the *ASCIIFLG* can be specified when starting a server to indicate that it should output the printed representation of the number. Similarly, when getting the time from the network, the following is used:

`RFC868.ASCII.OSTYPES = (VMS)` [Variable]

to decide based on the host's operating system whether to read the time as a binary or ASCII number. If this variable is set to `NIL`, the ASCII format is never used.

The ASCII format is currently only supported in the TCP protocol.



**PROTOCOL SPECIFIC FUNCTIONS**

(TCP.SETTIME *[RETFLG]*) [Function]

(UDP.SETTIME *[RETFLG]*) [Function]

Functions called by RFC868.SETTIME which can be called directly. The variables TCP.TIME.HOSTS and UDP.TIME.HOSTS must be set to use these functions.

(TCP.TIMESERVER *[ASCIIFLG]*) [Function]

(UDP.TIMESERVER) [Function]

Functions used by RFC868.START.SERVER. Can be used directly using ADD.PROCESS.

TCP.TIME.PORT = RFC868.TIME.PORT [Variable]

UDP.TIME.PORT = RFC868.TIME.PORT [Variable]

The ports to use in both the client and server functions.

TCP.TIME.HOSTS [Variable]

UDP.TIME.HOSTS [Variable]

Lists of host names and/or addresses (including broadcast addresses) to try to get the time from. Host are tried until one responds.

TCP.SETTIME.TIMEOUT = 10000 [Variable]

UDP.SETTIME.TIMEOUT = 10000 [Variable]

Length of time (in milliseconds) to wait for a host to respond to TCP.OPEN or UDP.EXCHANGE before trying the next one on the list.

---

---

**TEDITKEY**

---

---

By: Greg Nuyens

Supported by: Jan Pedersen (Pedersen.pa@Xerox.com)

Uses: KEYOBJ, DLIONFNKEYS

TEditKey is a module that provides a keyboard interface to TEdit. On a Dandelion, the interface takes advantage of the special keys to the left, top, and right of the main keyboard. On a Dorado or Dolphin, a window mimicking the Dandelion function keys provides some of the same abilities.

The abilities provided include allowing the user to alter the *caret looks* (the looks of characters typed in) or the selection looks. These commands are given using the Dandelion function keys and/or metacodes. (Metacodes are keys typed while a meta key is held down. The default meta key is the tab key; to alter this see "User Switches" below.) Other metacodes and control codes move the cursor within the document (beginning/end of line, back/forward a character, up/down a line, etc.).

Thus, many of the special Dandelion keys are made to function in TEdit the way they are labeled. The following keys change their behavior once TEditKey is loaded.

**CENTER** modifies the justification of the paragraph(s) containing the current selection. If the selection was left justified, then hitting the CENTER key makes it centered. Hitting it again produces right and left justification.

**BOLD** boldfaces the selection. All other properties remain unchanged. Holding the shift key down while hitting BOLD will make the selection become un-bold.

**ITALICS** italicizes the selection. Shift-ITALICS is the opposite.

**UNDERLINE** underlines the selection. Shift-UNDERLINE is the opposite.

**SUPERSCRIPT** superscripts the selection by a constant amount. Any relative superscripts (or subscripts) are maintained. Thus if "X<sub>i</sub>" is selected in "the set X<sub>i</sub> is empty" then pressing the SUPERSCRIPT button produces "the set X<sub>i</sub> is empty." See "User Switches" below for how to set the increment. Shift-SUPERSCRIPT is the same as SUBSCRIPT.

**SUBSCRIPT** is analogous to SUPERSCRIPT.

**SMALLER** decreases the font size of the selection. All relative size differences are maintained. E.g., "this is bigger than that" produces "this is bigger than that." Shift-SMALLER (labeled LARGER) does the opposite.

**DEFAULTS** makes the selection have default looks. N.B.: The default looks can be set to the current caret looks by typing shift-DEFAULTS.

The above keys all affect the caret looks if the keyboard key is held down when they are hit. Thus holding down KEYBOARD and then hitting UNDERLINE makes the caret looks be underlined.

**FONT** changes the font of the selection or caret looks according to the following table (to alter this table see "User Switches" below):

1	Times Roman
2	Helvetica
3	Gacha
4	Modern
5	Classic
6	Terminal
7	Symbol
8	Hippo

Thus, to change the font of the selection to Classic, hold down FONT and hit 5. To change the caret font to Classic, hold down FONT (to signal the font change) and KEYBOARD (to direct the change to the caret looks) then hit 5. Note that this table is part of the menu displayed when the HELP button is pressed.

On a Dorado, middle-blank is the FONT key.

**KEYBOARD** applies any changes that occur while this key is down to the caret looks instead of the selection. On a Dorado, bottom-blank is the KEYBOARD key.

**AGAIN** invokes the redo facility in TEdit. A wide variety of operations can be repeated very simply by making a selection, performing an operation (for instance, an insertion), then picking a new selection and hitting the AGAIN key. The AGAIN key is an ESCape key, which acts as the TEdit REDO syntax class. (See page 20.22 of the *Interlisp Reference Manual*.)

**OPEN** opens a blank line at the current cursor position. OPEN is also used to type a linefeed outside of TEdit (for example to the function FILES?).

**FIND** prompts the user for a target string, then searches from the selection forward.

**NEXT** acts as the TEdit NEXT syntax class. (It goes to the next field to be filled in. These fields are marked as follows: >>text to be substituted<<.)

**shift-NEXT** transfers the TTY (which window will receive typed characters) to the next window which can accept typein. Thus one can cycle through the open text windows (mail windows, top level lisp windows, TEdit windows, etc.) without using the mouse.

**EXPAND** expands TEdit abbreviations. (See page 20.31 of the *Interlisp Reference Manual*.)

**HELP** displays a menu of the keybindings until a mouse key is clicked.

**UNDO** acts as the TEdit UNDO syntax class. Note that it still retains its TELERAID function as does STOP. There are TEditKey operations (such as Transpose Characters) that are implemented with multiple TEdit operations. Since TEdit will UNDO only single operations, it does not fully UNDO these operations.

**RightArrow** enters \, and | when shifted. (Recall that AGAIN is an escape key.)

**MARGINS** indents the margins of the paragraph selected. Shift-MARGINS extends the margins. If the right margin is a floating margin, it is left unchanged. To control the amount by which the margins are moved, see "User Switches."

As well as the previous functions available on the Dandelion's special keys, the following functions are available on the standard keyboard (thus usable on the Dandelion, Dolphin, and Dorado). Each function is shown with the key that invokes it (in conjunction with the control (denoted ↑) or meta (denoted #) key). Thus, for the sixth entry, holding down the metakey and hitting f (or "F") would move the caret one word forward. (To find out how to get a metakey see "User Switches" below.)

#/	defaults the caret looks
# =	queries caret looks
#9	smaller caret font
#0	larger caret font
↑ b	back character
↑ f	forward character
#b	back word
#f	forward word
↑ p	previous line
↑ n	next line
↑ a	beginning of line
↑ e	end of line
#<	beginning of document
#>	end of document
#s	select whole document
↑ k	kills line (delete from caret to end of line)
↑ o	opens line
↑ d	deletes character forward (also on shift backspace)
#d	deletes word forward (as always ↑ w deletes word backward)
↑ t	transposes characters
#[	indents paralooks. Also available on the MARGINS key
#]	extends paralooks. Also available as shift-MARGINS
#j	justification change (same as CENTER key)
#u	uppercases selection
#c	capitalizes selection
#l	lowercases selection
#o	inserts object into document

#? shows keybindings (same as HELP)

#r restores the display

Note that the positions of any of these functions can be individually changed using TEDIT.SETFUNCTION (see page 20.30 of the *Interlisp Reference Manual*). For wholesale customization see "User Switches" below.

## INTERRUPTS

Any operation can be aborted by typing the STOP key. This can be used to abort font changes, GETs, PUTs, etc. A stronger form of interrupt is available as shift-STOP, which prompts the user for a menu of processes to interrupt.

↑G is available as a synonym for hitting the STOP key within TEditKey. Outside of TEdit, however, ↑G will continue to have the meaning specified in the user's init file. This is often the HELP interrupt, which acts as shift-STOP.

Users who are accustomed to typing ↑E as a soft interrupt should note that ↑E moves to the end of the line. As discussed above, hitting the STOP key (or equivalently, typing ↑G) accomplishes what ↑E did.

Since ↑H is defined to be the Backspace action in TEditKey, users cannot type ↑A to erase characters even outside of TEditKey (Interlisp-D currently does not allow multiple backspace characters).

In addition to the changed functionality mentioned above (provided courtesy of TEditKey), the user should be aware of the following standard Interlisp-D/TEdit behavior:

**SAME** operates as a LooksCopy mode key. First make a selection. Now to copy the looks from some other text simply hold down the SAME key, then select the source for the looks. (Paragraph looks can be copied the same way, but by making the final selection while in the left margin. This is the standard way to select a whole paragraph in TEdit.)

**MOVE** and **COPY** act as mode keys for the selection mechanism. Thus the user can select the destination, then hold down the MOVE key and make a second selection. This selection will be moved (or COPY'd depending on the mode key used) to the (original) caret position.

**CONTROL** operates as a mode key to signal deletion. This means that holding down the CONTROL key and selecting some text will delete that text when the CONTROL key is released.

**DELETE** deletes the current selection when pressed.

## DORADO EQUIVALENTS

Dandelion Key:	Equivalent key on Dorado:
<b>OPEN</b>	↑o (or ↑O)
<b>SAME</b>	META
<b>FIND</b>	finds item in TEdit menu
<b>AGAIN</b>	ESC
<b>DELETE</b>	DEL
<b>COPY</b>	SHIFT

<b>MOVE</b>	CTRL-SHIFT
<b>PROP'S</b>	META or LOCK depending on switches
<b>NEXT</b>	#n ( or #N)
<b>EXPAND</b>	↑ x (or ↑ X)
<b>HELP</b>	#?
<b>MARGINS</b>	#[ (unnest (which is shift-MARGINS on the Dandelion) is #] )
<b>FONT</b>	top blank
<b>KEYBOARD</b>	middle blank
<b>UNDO</b>	bottom blank
<b>STOP</b>	↑ G
<b>shift-STOP</b>	# ↑ S (intentionally difficult to type accidentally)

The function keys (CENTER, BOLD, etc.) are all available on the function key window brought up when TEditKey is loaded on a Dorado.

Note that the function key window can be rebuilt on a Dorado by selecting "Function Keys" in the default TEdit menu (obtained by buttoning in the title bar of a TEdit window).

#### USER SWITCHES

**TEDITKEY.METAKEY** The user must choose a metakey to make use of TEditKey. The value of the variable **TEDITKEY.METAKEY** is the name of the key that will be your metakey. For instance to make TAB (the default) your metakey, (SETQ TEDITKEY 'TAB) before loading TEditKey. (Note that even in the standard system, TAB is available as Control-I).

NOTE: METASHIFT (see page 18.9 of the *Interlisp Reference Manual*) is redefined to operate on TEDITKEY.METAKEY instead of on the bottom-blank key of the Dorado.

The operation of TEditKey is controlled by the following (INITVARed) variables:

**TEDITKEY.LOCKTOGGLEKEY** is the key that will be turned into a lock-toggle. If it is non-NIL, that key is set to act as a lock-toggle. Thus hitting this switches the case of the type-in. For those users who have removed the spring from their lock key, **TEDITKEY.LOCKTOGGLEKEY** is usually PROP'S. The action of LOCK is then made to be '(CTRLDOWN. CTRLUP) providing the user with a control key where LOCK is located and a lock toggle where PROP'S is located.

**TEDITKEY.FONTS** is an eight-element list of the fonts that are invoked by meta-1 through meta-8. The defaults are listed above.

**TEDIT.DEFAULT.CHARLOOKS** defines the looks that result when the DEFAULTS key is pressed or when default caret looks are requested. It is an instance of the CHARLOOKS datatype. To preset it, for instance, to TIMESROMAN 10 type the following to the Lisp top level.

```
(SETQ TEDIT.DEFAULT.CHARLOOKS (CHARLOOKS.FROM.FONT (FONTCREATE 'TIMESROMAN 10)))
```

However, a much simpler method is to select an instance of the desired looks and type shift-DEFAULTS.

**TEDITKEY.VERBOSE** if T (the default), the functions that modify the caret looks print feedback in the (TEdit) prompt window.

**TEDITKEY.NESTWIDTH** is the distance (in points) that the indent and extent functions move the margins. Initially 36 points (0.5 inches).

**\TK.SIZEINCREMENT** is the amount (in points) which the LARGER function increases the selection (and conversely for SMALLER). Initially 2 points.

**TEDITKEY.OFFSETINCREMENT** is the amount (in points) which the SUBSCRIPT function raises the selection (and conversely for SUPERSCRIPT). Initially 3 points.

**TEDITKEY.KEYBINDINGS** is the list that controls the mapping of keys to functions for the functions that are common to the Dandelion, Dorado, and Dolphin. It consists of triples of function name, list of CHARCODE-style character specifications, and a comment describing what the function does. (The comments are used by the automated menu-building tools and their inclusion is encouraged.)

**TEDITKEY.DLION.KEYACTIONS** is the list that specifies the key actions of the non-Alto keys (to the left and right) on the Dandelion. It is the format acceptable to MODIFY.KEYACTIONS (see page 18.9 of the *Interlisp Reference Manual*).

**TEDITKEY.DLION.KEYBINDINGS** is the list specifying the functions to be tied to the characters generated from above. The keynames in the CAR of each element are comments. Note that TEDIT.DLION.KEYACTIONS and TEDIT.DLION.KEYBINDINGS must be coordinated (similarly for TEDITKEY.FNKEYACTIONS and TEDITKEY.FNKEYBINDINGS).

**TEDIT.DLION.KEYSYNTAX** is the list of syntax classes to be applied to the Dandelion keys.

**TEDITKEY.FNKEYACTIONS** is the list that specifies the keyactions of the function keys (center, bold, etc.).

**TEDITKEY.FNKEYBINDINGS** is analogous to TEDIT.DLION.KEYBINDINGS but for the function keys.

**TEDITKEY.DORADO.KEYACTIONS** are the keyactions unique to the Dorado (and Dolphin).

**TEDITKEY.DORADO.KEYSYNTAX** is analogous to TEDIT.DLION.KEYSYNTAX.

The previous variables in conjunction with the following functions specify the effect of TEditKey.

**(TEDITKEY.INSTALL *readtable*)** invokes the keyactions and bindings as specified by the above variables on *readtable*. (*Readtable* defaults to TEDIT.READTABLE).

**(\TK.BUILD.MENU)** is a function that automagically builds the help menu from the values of the above variables.

---



---

## TILED-SEDIT

---



---

By: Johannes A. G. M. Koomen  
(Koomen.wbst@Xerox or Koomen@CS.Rochester)

This document last edited on: September 23, 1987

### SUMMARY

TILED-SEDIT is a facility for automagically positioning SEdit windows according to a specified pattern. SEdit windows appear in any of the four corners of the screen, with overlapping windows slightly offset so they can still be brought to top (by clicking on them). Users can specify which corners, in what order, how thick a margin around the screen, and the size of the offset.

### DESCRIPTION

(TILED.SEDIT.RESET *Tiling-Order XShift YShift Screen*) [Function]

If *Tiling-Order* is NIL, this resets the SEdit window tiling facility, and SEdit reverts back to its old behavior (i.e., prompting for a window region). Otherwise *Tiling-Order* should be either T or a keyword or an arbitrarily long list of keywords from the following set { :TL :TOP-LEFT :TOP.LEFT :TOPLEFT :BL :BOTTOM-LEFT :BOTTOM.LEFT :BOTTOMLEFT :TR :TOP-RIGHT :TOP.RIGHT :TOPRIGHT :BR :BOTTOM-RIGHT :BOTTOM.RIGHT :BOTTOMRIGHT }. If *Tiling-Order* is T, the list (:TL :BL :TR :BR) is assumed. SEdit will place new windows in the corners specified by *Tiling-Order* (which is indefinitely repeated if necessary).

If a new SEdit window would overlap an existing SEdit window, the new one is offset by *XShift* pixels right and *YShift* pixels down. *XShift* and *YShift* default to 15. Tiled.SEdit will compute the tile size and placement on the basis of the region *Screen* such that you can go three times through the default four corner loop before the right or bottom windows start crossing the edge of *Screen*. If *Screen* is neither a region nor a fixp, *Screen* defaults to 25. If *Screen* is a fixp M, *Screen* is assumed to be (CREATEREGION M M SCREENWIDTH-M SCREENHEIGHT-M). The default setting leaves room enough for a scrollbar on the left and the bottom.

Invoking TILED.SEDIT.RESET with a non-NIL *Tiling-Order* will cause all currently open SEdit windows to be repositioned according to *Tiling-Order*.

### EXAMPLES

(TILED.SEDIT.RESET T) [Function]

This is executed when you load TILED-SEDIT. It provides for automatic SEdit window creation in the corners TopLeft, BottomLeft, TopRight, BottomRight, TopLeft, BottomLeft, ... Each time around the loop windows are shifted 15 pixels to the right and downward. A 25 pixels margin is preserved at the left and bottom edge of the screen.

(TILED.SEDIT.RESET :TL) [Function]

This causes SEdit to create windows in the TopLeft corner only. Each new window is shifted 15 pixels to the right and downward. A 25 pixels margin is preserved at the left and bottom edge of the screen.



(TILED.SEDIT.RESET '(:TR:BR) NIL 35)

[Function]

This causes SEdit to create windows in the TopRight and BottomRight corners only. Each time around the two corner loop windows are shifted 15 pixels to the right and 35 pixels downward. This has the advantage that the title of each SEdit window remains visible, but the disadvantage that each window is smaller. A 25 pixels margin is preserved at the left and bottom edge of the screen.

#### CAVEAT

TILED.SEDIT.RESET is independent of SEDIT.RESET. It will not invoke SEDIT.RESET, nor does it require that all SEdit windows are closed prior to invocation. It is strictly used for controlling the window tiling.

---



---

## Trajectory-Follower

---



---

By: D. Austin Henderson, Jr. (AHenderson.pa@Xerox.com)

### INTRODUCTION

Trajectory-Follower provides a function which causes a "snake" to crawl along a trajectory. Comments on both interface and functionality are welcomed.

### FUNCTIONS

(TRAJECTORY.FOLLOW *KNOTS CLOSED N DELAY BITMAP WINDOW*) [Function]

The trajectory is specified by *KNOTS* (a set of knots) and *CLOSED* (a flag indicating whether it is an open or closed curve). *N* is the length of the snake in points along the curve. *DELAY* is the time (in milliseconds) between each move along the curve; *DELAY* = 0 or NIL means go as fast as you can. *BITMAP* is the brush to be used at each point in creating the snake. *WINDOW* is the window in whose coordinate system the knots are given and in which the snake is to be drawn; if NIL, then the SCREEN bitmap is used. The snake is moved by INVERTing the bitmap at the points along the curve, and then INVERTing the bitmap back out again.

#### Examples

A demonstration function is also provided with the module:

(TRAJECTORY.FOLLOWER.TEST) [Function]

Interacts with the user through prompting in the promptwindow to gather up arguments for TRAJECTORY.FOLLOW and then carries it out. Closed curves are snaked around repeatedly until the left shift key is found depressed when it reaches the curve's starting point.

#### Internal Functions

The internal functions used by this module are also available for use. They are:

(TRAJECTORY.FOLLOWER.SETUP *WINDOW N DELAY BITMAP*) [Function]

Initializes drawing variables.

(TRAJECTORY.FOLLOWER.POINT *X Y WINDOW*) [Function]

Defines the next point on the curve. Note that the argument structure of this function is appropriate for use as a BRUSH with the curve drawing functions DRAWCURVE, DRAWCIRCLE, and DRAWELLIPSE. (For an example, see the demonstration function TRAJECTORY.FOLLOWER.TEST)

(TRAJECTORY.FOLLOWER.WRAPUP) [Function]

Finishes the job after all the points have been defined.

---



---

**TRICKLE**


---



---

By: Nick Briggs (Briggs.pa@Xerox.com)

Uses: PROMPTREMINDERS

This document last edited on October 12, 1987

## INTRODUCTION

Trickle provides a very simple cover for COPYFILES to do periodic (every 24 hours) updating of one directory from another, with processing of the log files generated by COPYFILES to mail a note to some designated person indicating what COPYFILES did.

## USE

There is only one function of interest to the user:

(Trickle *Source Destination RootLogfileName MailAddress ScheduleAnotherOne*  
*DontReplaceOldVersions*) [Function]

*Source* and *Destination* should be patterns acceptable to COPYFILES. *RootLogfileName* should be a host, directory, and partial file name to which Trickle will append the date in the form yymmdd, and the extension .CopyLog. On completion of the copy Trickle will mail a message to *MailAddress* if it is non-NIL. If *ScheduleAnotherOne* is T then another Trickle will be scheduled (randomly) between 1 am and 5:59 am of the next day, alternatively, if *ScheduleAnotherOne* is a time that would be acceptable to IDATE (Trickle will prepend the actual date, just give the time) then another Trickle will be scheduled at exactly that time. *DontReplaceOldVersions* signals that Trickle should not use the COPYFILES option REPLACE, use of which causes problems with NS file servers (at least in Koto).

## Example

To update the directory {cf}<lispusers>koto>\* from {eris}<lispusers>koto>\* storing the log files starting with {core}eluk-870512.copylog, mailing notification to Briggs.pa, scheduling this to run every night, and using COPYFILES' REPLACE option one would execute:

```
(SETREMINDER NIL NIL
  '(Trickle '{eris}<lispusers>koto>* '{cf}<lispusers>koto>*
    "{core}eluk-" "Briggs.pa" T)
  "12-May-87 03:00")
```

Two versions of the log file will be created; version 1 with the complete log output of COPYFILES, and version 2, with all the "skipped" files removed. It is this version that is mailed to the designated recipient.

The mail messages that are sent out indicate whether there were any files processed: the subject line will include the string "(Empty)" if no files were Trickled, and the string "(Error?)" if there were no files in the source directory (may not be an error, but may be worth investigating)

---

---

**TURBO-WINDOWS**

---

---

By: Andrew J. Cameron, III (Cameron.pa@Xerox.com or cameron@cs.wisc.edu)  
New Owner: Atty Mullins (Mullins.pa@Xerox.com)

Uses: WDHACKS (LispUsers) [optional]

This document last edited on Sept. 8, 1988.

### INTRODUCTION

Turbo-Windows does not have anything to do with speeding up primitive window operations, but rather it helps speed up your use and manipulation of windows by providing most of the right button menu functions via shift keychords. In this way one can Move, Shape, Copy, Shrink, Close, etc., a window without having to wait for the right button menu to appear and then select from it.

Also, when providing the INITIAL shape of a window, pressing the middle button yields a large default size suitable for TEdit, etc. (Recall that using the middle button during a RESIZING operation allows you to keep roughly the original window shape and then move the corner nearest the cursor when the middle button was pressed.)

One can bring up a brief cribsheet for all the TurboWindow keychord commands by holding down the HELP key and RIGHT buttoning on the background (not in any window). This can also be produce by typing (TW.HELP) to an InterLisp EXEC.

### OPERATION

Before discussing how to use this utility, a description of how the key-chords were chosen is in order. They are based loosely on the effect of the shift keys in TEdit. Recall that in TEdit, pressing and holding the Shift key Copies whatever is selected. Also, pressing and holding the Control key (sometimes labeled PROPS or EDIT) Deletes whatever is selected. Pressing both Shift and Control performs both a copy and a delete, which ends up Moving the selected item. The only additional piece of information that you need to know is that the Meta key (sometimes labeled KEYBOARD or ALT) modifies an operation or in some way makes it different. With this general interpretation, most of the key-chords are rather easy to remember.

If the following keys are chorded (held down together) while the right mouse button is pressed in the region of a window which would normally bring up the right-button menu (by convention, at least the title bar should provide the right button menu), the listed operation will be invoked without actually bringing up the right button menu.

**SHIFT** (using the LEFT SHIFT key or CAPS LOCK key)

Makes a copy of a window by snapping it.

### CONTROL

Closes (deletes) a window. (Since this is a destructive operation, a small safeguard is built into this operation. If one holds the CONTROL key and depresses the right mouse button and continues to

hold them, the window to be operated on (closed) will blink. If this is not the window you want to close you can cancel the Turbo-Close by either moving outside the window (or by releasing the CONTROL key before releasing the right mouse button). If you abort the Turbo-Close in this manner, the normal right button menu will appear. Clicking outside of the menu will make it go away. Sometimes unexpected things occur when trying to Turbo-Close windows with attached windows, e.g. FileBrowsers, but hopefully this safeguard is conservative enough to avoid inadvertent closing of the wrong window.) [Holding down CONTROL while Right Buttoning on the background activates Window Slamming, if the LispUsers utility WDHACKS is loaded.]

#### META

Shape (makes different) a window.

#### SHIFT-CONTROL

Moves a window. (Due to the design of the InterLisp window system, this operation works in a rather strange way. You press and hold both CONTROL and SHIFT and then press the right mouse button while in the appropriate part (title bar) of the window you want to move. You then need to release the right button to be able to actually move the window. In order to "drop" the window (here is the strange part) you need to press the LEFT (or middle) button. Pressing the right button merely allows you to move to a different corner of the shadow box.)

#### META-CONTROL

Shrinks ("deletes" in a different way) a window.

#### META-SHIFT

Redisplays (copies in a different way) a window.

#### META-SHIFT-CONTROL

Buries (moves in a different way) a window. [You might also think of this as pushing the window down to the bottom, as you are pressing down all three shift keys.]

#### RIGHT-SHIFT

Clears a window.

#### HELP

Pressing the HELP key while the cursor is in the background (or typing (TW.HELP) to an InterLisp EXEC) displays a crib sheet for the Turbo-Window KeyChords. Some additional capabilities not listed here are given on that crib sheet. The "OTHER" keychords which are marked with an asterisk (\*) indicate that some side-effect (potentially quite harmful) might occur depending on where the TTY is when those alternate access methods are used. You are warned!

#### GETTING STARTED

[If any of the operations described below do not perform properly, it might be the case that your keys are not defined in the way that this utility expects. See INTERNALS below for more information.]

You might want to get familiar with Turbo-Windows by first bringing up the crib sheet by depressing the HELP key and right-buttoning on the background. Next, make a copy of the crib sheet by depressing SHIFT (the left shift key) and right-buttoning on the crib sheet. Drop the new copy of the crib sheet by releasing all keys and buttons and the pressing the left mouse button. [Note: The crib sheet is merely written to the TTY window, which happens to be sensitive to the

right mouse button everywhere. Other windows may only be sensitive to the right button (for the purpose of bringing up the right button menu, in their title bar.) Now try moving the copied cribsheet by pressing both SHIFT and CONTROL (PROPS or EDIT) and right-buttoning on the copy of the cribsheet. Again, release everything (well, just releasing the mouse button will do) and press the left mouse button to drop it. Press and hold both META (KEYBOARD) and CONTROL while right buttoning in the copy of the cribsheet to shrink it to an icon. Release and click the left mouse button to drop the icon. Reopen (expand) the icon by middle buttoning on it. Reshape the copy of the cribsheet by pressing META and right buttoning on the copy's window. Release and rubberband the new shape with the left mouse button. (Do you know what would happen if you used the middle button after releasing instead? Try it.) Assuming the copy of the cribsheet is overlapping another window and some part of the background (if not, Turbo-Move it so it is), press and hold all three (META, SHIFT, and CONTROL) and right button in the cribsheet copy's window to bury it. Right button (holding no other keys) in the partially exposed area of the now buried cribsheet copy to bring it back to the top. Finally, close the copied cribsheet window by pressing CONTROL while right buttoning in the copy's window. [O.K. which shift key combination hasn't been used yet? Consult the original cribsheet (or produce it again), if necessary. Give that combination a try in the original cribsheet's window. [Did you notice the message in the prompt window?] And don't forget to give the Right Shift key (Clears a window) a try as well. [Remember, the cribsheet can be brought back at any time using HELP-RightButton on the background.] ) To see how to cancel a Turbo-Close, depress the CONTROL key and press AND HOLD the right mouse button while in the original cribsheet window. Notice that the window blinks. Before you release the right mouse button move the cursor outside the cribsheet's window and then release the right mouse button. The cribsheet's window is not closed because releasing outside the window that flashed cancels the Turbo-Close. The normal right button menu appears instead. Click outside it to get rid of it. Now, actually close the original cribsheet window. And with that, may I welcome you to the fast paced world of Turbo-Windows.

## INTERNALS

The right button events are intercepted by a piece of advice placed on DOWINDOWCOM. The middle button sizing capability is provide by advice on \GETREGIONTRACKWITHBOX.

The window snapping Turbo-Window feature (LeftShift-RightMouseButton) is also added as a submenu to the normal right button menu provided by the window system.

A common problem is that the META key is not defined to be at the proper place (attached to the key named KEYBOARD). To remedy this, type:

```
(KEYACTION 'KEYBOARD '(METADOWN . METAUP))
```

to an InterLisp EXEC. The following should also be the case:

```
(KEYACTION 'EDIT '(CTRLDOWN . CTRLUP))
```

```
(KEYACTION 'LSHIFT '(1SHIFTDOWN . 1SHIFTUP))
```

```
(KEYACTION 'RSHIFT '(2SHIFTDOWN . 2SHIFTUP))
```

These can be verified by using, for example:

```
(KEYACTION 'EDIT)
```

TW.NO-FLASH-CLOSE

[Variable]

Initially NIL, if set to T, windows will not flash to indicate there impending closure.

TW.DONT-GROW-SNAP-BORDER [Variable]

Initially NIL, if set to T, windows will be copied without a small border. The small border is quite handy in telling the original window from its Turbo-Snapped copy.

TW.SNAP-HERE [Variable]

Initially NIL, if set to T, windows will be copied directly on top of the window they are duplicating. Normally (when NIL) the user must position the copy.

GETREGIONDEFAULT [Variable]

This variable can be bound dynamically by an application to provide the region afforded by middle buttoning when prompted for an initial region of a window. It is initially set to roughly 7x9 inches, and is useful for TEdit windows, FileBrowsers, etc. [See the LispUsers utility RESIZE-FILEBROWSER for an even better way of dealing with FileBrowsers.]

- In order to edit/compile the source of this utility, the InterLisp Source file WINDOW must be loaded in order to provide the SCREEN record definition used by the window system internals. The loading of this source file occurs automatically when this utility's source file is loaded.
- This utility interacts poorly with other utilities that redefine any of the shift keys. TEDITKEY and PC-Emulation (among others) are dubious in this regard.

---



---

**TWODGRAPHICS**


---



---

By: Jan Pedersen (Pedersen.PA @ Xerox.com)

Uses: UNBOXEDOPS

TWODGRAPHICS implements viewports. A viewport is a subregion of a window (or image stream) within which graphics is clipped and a linear transformation from a world coordinate system to the window (or image stream) coordinates.

A given window (or image stream) may have any number of viewports defined and the viewports may be arbitrarily nested or overlapping. If a window is reshaped the subregions of all currently defined viewports are proportionately reshaped.

Viewports will operate in the context of any image stream, (Interpress printers, etc.), although not all DIG (Device independent graphics) primitives are supported.

(CREATEVIEWPORT *stream streamsubregion source*) [Function]

Creates a viewport on stream. Stream is the target stream. Streamsubregion is a region in stream coordinates that defines the extent of the viewport.

Source may be a REGION in world coordinates, in which case the world to stream linear transformation is set up to map left to left and bottom to bottom, etc., or a VIEWPORT, in which case the new viewport inherits its world to stream transformation.

Returns a VIEWPORT

(SETWORLDREGION *region viewport*) [Function]

(Re)sets the worldregion of viewport and recomputes the transformation.

(SETSTREAMSUBREGION *region viewport*) [Function]

(Re)sets the streamsubregion of viewport and recomputes the transformation.

Modified versions of selected DIG primitives are supplied to take advantage of the world to stream transformation.

(TWODGRAPHICS.BITBLT *source sourceleft sourcebottom destinationviewport  
destinationleft destinationbottom width height  
sourcetype operation texture clippingregion*) [Function]

World coordinates may be used where it makes sense. The destination must be a VIEWPORT. Destination left and bottom default to the viewport's stream subregion left and bottom. The clippingregion argument is always in destinationviewport world coordinates. The source may be a VIEWPORT, a BITMAP, or NIL in the case of texture patterns.

In the following, all coordinates must be world coordinates.



(TWODGRAPHICS.MOVETO <i>x y viewport</i> )	[Function]
(TWODGRAPHICS.MOVETOPT <i>position viewport</i> )	[Function]
Here position is a POSITION in world coordinates	
(TWODGRAPHICS.RELMOVETO <i>dx dy viewport</i> )	[Function]
(TWODGRAPHICS.RELMOVETOPT <i>dposition viewport</i> )	[Function]
(TWODGRAPHICS.DRAWTO <i>x y width operation viewport color dashing</i> )	[Function]
(TWODGRAPHICS.DRAWTOPT <i>position width operation viewport color dashing</i> )	[Function]
(TWODGRAPHICS.RELDRAWTO <i>dx dy width operation viewport color dashing</i> )	[Function]
(TWODGRAPHICS.RELDRAWTOPT <i>dposition width operation viewport color dashing</i> )	[Function]
(TWODGRAPHICS.DRAWLINE <i>x1 y1 x2 y2 width operation viewport color dashing</i> )	[Function]
(TWODGRAPHICS.DRAWBETWEEN <i>position1 position2 width operation viewport color dashing</i> )	[Function]
(TWODGRAPHICS.DSPRESET <i>viewport</i> )	[Function]
Does a "DSPRESET" on the VIEWPORT	
(TWODGRAPHICS.DSPFILL <i>region texture operation viewport</i> )	[Function]
region must be in world coordinates	
The following function is an extension which may be of use to those who wish to produce analytic plots.	
(TWODGRAPHICS.PLOTAT <i>position glyph viewport operation</i> )	[Function]
Bitblts glyph to position with operation, with glyph centered at position.	
Several functions provide access to the world to stream transformations.	
(WORLDTOSTREAM <i>position viewport oldposition</i> )	[Function]
Position is in world coordinates. Oldposition is smashed if provided.	
Returns the corresponding position in stream coordinates.	
(WORLDREGIONTOSTREAMREGION <i>region viewport</i> )	[Function]
Region is in world coordinates	
Returns the corresponding region in stream coordinates	
(WORLDTOSTREAMX <i>x viewport</i> )	[Macro]
Returns x in stream coordinates.	
Uses unboxed floating point arithmetic	
(WORLDTOSTREAMY <i>y viewport</i> )	[Macro]
Returns y in stream coordinates	
Uses unboxed floating point arithmetic.	

- (WORLDXLENGTH *dx viewport*) [Macro]  
Returns the length dx in stream coordinates  
Uses unboxed floating point arithmetic.
- (WORLDYLENGTH *dy viewport*) [Macro]  
Returns the length dy in stream coordinates.  
Uses unboxed floating point arithmetic.
- (STREAMTOWORLD *position viewport oldposition*) [Function]  
Returns position in world coordinates.
- (STREAMTOWORLDX *x viewport*) [Macro]  
Returns x in world coordinates.  
Uses unboxed floating point arithmetic.
- (STREAMTOWORLDY *y viewport*) [Macro]  
Returns y in world coordinates.  
Uses unboxed floating point arithmetic.
- (STREAMXLENGTH *dx viewport*) [Macro]  
Returns dx in world coordinates.  
Uses unboxed floating point arithmetic.
- (STREAMYLENGTH *dy viewport*) [Macro]  
Returns dy in world coordinates.  
Uses unboxed floating point arithmetic.
- For those who desire tighter control over the two-stage process, transform into stream coordinates, and then clip against the viewport, the following functions provide primitive clipping for line drawing and text output in any image stream.
- (CLIPPED.BITBLT *clippingregion source sourceleft sourcebottom  
destination destinationleft destinationbottom  
width height sourcetype operation texture*) [Function]  
As in BITBLT, although the operation is clipped against clippingregion in destination stream coordinates.
- (CLIPPED.DRAWLINE *clippingregion x1 y1 x2 y2 width operation stream  
color dashing*) [Function]  
As in DRAWLINE, although the operation is clipped against clippingregion in stream coordinates.
- (CLIPPED.DRAWTO *clippingregion x y width operation stream color dashing*) [Function]  
As in DRAWTO, although the operation is clipped against clippingregion in stream coordinates.

(CLIPPED.DRAWBETWEEN *clippingregion pt1 pt2 width operation stream color dashing*) [Function]

As in DRAWBETWEEN, although the operation is clipped against clippingregion in stream coordinates.

(CLIPPED.PLOTAT *clippingregion position glyph stream operation*) [Function]

BITBLT glyph to stream centered at position and clipped against clippingregion.

(CLIPPED.PRIN1 *clippingregion expr stream*) [Function]

PRIN1 expr on stream clipped against clippingregion.

---



---

## UNBOXEDOPS

---



---

By: Jan Pedersen(Pedersen.PA @ Xerox.com] and Larry Masinter (Masinter.PA @ Xerox.com]

The module UNBOXEDOPS is intended to assist those interested in high-performance, scalar, floating-point arithmetic. The basic trick is to perform floating point arithmetic on the stack, utilizing special, unboxed, floating-point opcodes, an ugly but usually effective solution. This method of eliminating floating-point number boxes is likely to change, but in the interim a combination of compiler declarations and explicit evocations of unboxed operations, as described below, will allow the interested user to eliminate a high percentage of floating-point number boxes. This module and the methods described are "safe", i.e., the declarations won't cause your programs to crash, and if it works with the declarations it will also work without them.

Unboxed floating point tricks help out only 1108's with floating point hardware or 1186's with floating point microcode. Unfortunately, they may make performance even worse on 1108's without floating point hardware, although the performance degradation is probably not too severe.

There exist opcodes which perform floating point arithmetic on the stack (that is, on the bits of those numbers, rather than pointers to those bits). These opcodes are only emitted by the byte compiler if arithmetic occurs in an unboxed context. One example of an unboxed context is arithmetic on a record field defined to be of type FLOATP, another is arithmetic on a variable declared to be of TYPE FLOAT . However, the compiler will box across function boundaries and in a return context. Furthermore, there exist more unboxed opcodes than are used by the compiler (unboxed comparison springs to mind).

UNBOXEDOPS defines macros/functions so that these additional opcodes may be exploited in an unboxed context. These macros/functions include:

UFABS, UFEQP, UFGEQ, UFGREATERP, UFIX, UFLEQ, UFLESSP, UFMAX, UFMIN, UFMINUS, and UFREMAINDER,

which behave identically to there non-U namesakes, except that the operations are done on the stack without generating floating point boxes.

For those unfamiliar with unboxed compiler declarations a short description follows:

**Using (DECLARE (TYPE FLOATING x y z)) to reduce number boxes**

Consider the silly function:

```
(DEFINEQ (FIE (N)
  (bind (SETQ X 0.0) (SETQ Y 2.0) for I from 1 to N
    do (SETQ X (FPLUS X (FTIMES Y Y)))
    finally (RETURN X))))
```

```
(TIMEALL (FIE 100))
```

returns a CPU time of .025 and reports 200 FLOATP boxes produced. Now, consider

```
(DEFINEQ (FOO (N)
  (bind (SETQ X 0.0) (SETQ Y 2.0) for I from 1 to N
    declare (TYPE FLOAT X Y)
    do (SETQ X (FPLUS X (FTIMES Y Y)))
    finally (RETURN X))))
(TIMEALL (FOO 100))
```

returns a CPU time of .003 seconds and reports just one floatp box produced.

Essentially the (TYPE FLOAT X Y) declaration is a promise to the compiler that X and Y will hold FLOATP's, so arithmetic may be done unboxed (that is on the value itself, instead of on a pointer to the value, which is the usual case) if possible. The key issue is what is meant by "if possible".

The compiler is conservative. It will perform unboxed arithmetic only on built-in arithmetic functions (PLUS, TIMES, DIFFERENCE, etc), which have unboxed counter parts, and will otherwise box across function boundaries regardless of compiler declarations.

For example:

```
(DEFINEQ (FOOBAR (N)
  (bind (SETQ X 0.0) (SETQ Y 2.0) for I from 1 to N
    declare (TYPE FLOAT X Y)
    do (SETQ X (FPLUS X (LOG Y)))
    finally (RETURN X))))
```

then

```
(TIMEALL (FOOBAR 100))
```

returns a CPU time of .049 with 601 FLOATP boxes produced (some of which come from the LOG (five per function call)).

Also, the compiler will box in a return context. For example

```
(DEFINEQ (BAR (N)
  (bind (SETQ X 0.0) for I from 1 to N
    declare (TYPE FLOAT X )
    do (SETQ X
      (PROG ((Y 2.0))
        (DECLARE (TYPE FLOAT Y))
        (RETURN (FTIMES Y Y))))
    finally (RETURN X))))
```

then

```
(TIMEALL (BAR 100 ))
```

returns a CPU time of .022 with 301 FLOATP boxes produced -- notice that BAR seems like it should behave like FOO.

Indeed that is the the greatest drawback of the unboxed arithmetic as it stands now -- it is not always easy to predict what is going to happen -- there are even traps where indiscriminate uses of TYPE FLOAT declarations will actually produce MORE boxes than without them. This is the case if,

for example, you use comparison operators (GREATERP, etc) since the compiler boxes each operand before invoking them.

The BAR example may be fixed up as follows:

```
(DEFINEQ (BAR (N)
  (bind (SETQ X 0.0) for I from 1 to N
    declare (TYPE FLOAT X )
    do (SETQ X
      (PROG ((Y 2.0) RESULT)
        (DECLARE (TYPE FLOAT Y RESULT))
        (RETURN (SETQ RESULT (FTIMES Y Y))))
    finally (RETURN X)))
```

then

```
(TIMEALL (BAR 100))
```

returns a CPU time of .008 with 101 FLOATP boxes produced. Note that the compiler still boxes the result returned by the PROG.

The best way to find out what is happening is to use a combination of TIMEALL and INSPECTCODE . Unanticipated boxing behavior will show up as BOX opcodes -- if you find a sequence of opcodes UNBOX , BOX , function call, UNBOX , then you know you are in trouble. TIMEALL will report the total number of boxes produced.

Basically TYPE FLOAT declarations are best used in tight inner loops of the sort illustrated in FOO.

With all these caveats, I think it is only fair to say that considerable performance improvements can be realized with judicious use of the TYPE FLOAT declarations; my measurements indicate a factor of ten.

Additional note: TYPE FLOAT vars are by necessity LOCALVARS.

Lyric compatibility note: All the entries described for this module are in the Interlisp package. Only the Byte compiler pays attention to TYPE FLOAT declarations -- i.g. use of TYPE FLOAT declarations will be ignored by the XCL compiler.

---

---

**UUENCODE**

---

---

By: Doug Cutting (Cutting.PA@Xerox.COM)

This document last edited on October 7, 1987.

UUENCODE provides facilities for encoding files into printing ASCII characters for transfer by electronic mail. It is compatible with the UNIX™ facility of the same name. For details of the file format see the UNIX™ manual page on 'uencode'.

**(UUENCODE FILES INTO-FILE)** [Function]

Encodes the files named by FILES into INTO-FILE. FILES may be either a list of files or a single file name. Note that UNIX™ udecode does not support multiple files encoded in one file. Thus one should only pass a single file name to UUENCODE if the file is to be decoded under UNIX™. Returns the name of the file written.

**(UUDECODE FILE-OR-STREAM ONLY-ONE-FILE?)** [Function]

Decode from FILE-OR-STREAM writing the decoded files in the connected directory. FILE-OR-STREAM may be either a file name or a stream. If ONLY-ONE-FILE? is non-NIL then only one file will be extracted from FILE-OR-STREAM, and an error will be reported if no encoded file is found. This can be thought of as UNIX™ compatibility mode. Returns the list of the names of the files extracted.

**(UUENCODE-INTERNAL INS OUTS DECODE-NAME FILE-MODE)** [Function]

Called by UUENCODE to encode one file. Encodes all bytes from the stream INS to the stream OUTS. DECODE-NAME is the name the file should be given when it is decoded. FILE-MODE is the UNIX™ file mode for the file. DECODE-NAME defaults to (FULLNAME INS) and FILE-MODE defaults to the value of the variable UU.MODE-DEFAULT. Returns OUTS.

**UU.MODE-DEFAULT** [Global Variable]

The default UNIX™ file mode to encode files under as an integer. UNIX™ udecode will use this when creating the decoded file. The initial value is 644Q (read & write by owner, read by group and other).

**(UUDECODE-INTERNAL INS ONE-FILE-ONLY?)** [Function]

Called by UUDECODE to decode one file. INS should be a stream open for input. Returns the name of the file extracted or NIL if none is found and ONE-FILE-ONLY? is NIL.

UUENCODE was inspired by Christopher Lane's BMENCODE package.

---



---

## VSTATS

---



---

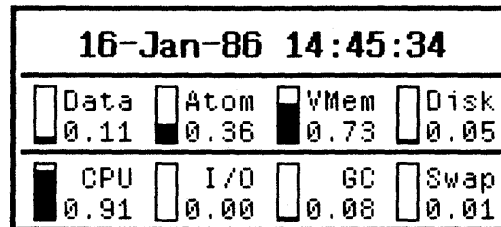
By: Johannes A. G. M. Koomen  
(Koomen.wbst@Xerox or Koomen@CS.Rochester)

Uses: SYSTATS, READNUMBER

This document last edited on November 20, 1987

### INTRODUCTION

Loading VSTATS will put a VStats entry on the background menu, and execute (VSTATS 'On), which will cause the following display to be created and continuously updated:



### DESCRIPTION

VSTATS is a facility for continuously displaying various interesting aspects of a running system. It can display the current time of day, with or without seconds, and/or display memory and disk space utilization, and/or display machine utilization in terms of CPU, I/O, GC and swap time. The display can be regular or inverse-video. The display is updated at user settable intervals, either always or only if the display window is completely visible.

Closing the VSTATS window will remove the background update function.

Left buttoning the VSTATS window causes it to be recomputed and redisplayed entirely. Otherwise display updates only affect those parts that have actually changed, making for a visually quiet and efficient facility.

Middle buttoning the VSTATS window will bring up an Inspector window onto the VSTATS list of options. Left buttoning an option name prints an explanation of the option to the Prompt window. Left buttoning an option value selects it, and middle buttoning an option value presents a menu from which a new value can be selected. The options window looks like this:



VStats Options	
Display.Color	Normal
Update.Always?	No
Show.Disk.Space?	<b>DSK1</b>
MUtil.Hysteresis	20
Space.Panic.Level	Disabled
Clock.Update.Interval	1
Space.Update.Interval	300
MUtil.Update.Interval	1

VSTATS is highly optimized for speed and implemented as a BACKGROUNDFN rather than as a seperate process so as to minimize overhead. As a result, VSTATS can easily be run with display update intervals equal to 1 second.

**DETAILS**

(VSTATS *on/off*) [Function]

If *on/off* is either ON, On, on, or T, the VStats display is turned on; otherwise off.

VSTATS.CLOCK.INTERVAL [Variable]

If the global variable VSTATS.CLOCK.INTERVAL is a positive number, VSTATS displays an alphanumeric clock (e.g., "1-Aug-85 14:30"), which is updated every VSTATS.CLOCK.INTERVAL seconds. If this interval is less than 1 minute VSTATS displays seconds as well. For those of you who keep their machines running overnight (say, with IDLE or BOUNCE), if the clock display is enabled, VSTATS will resynchronize the local clock with the network daily at midnight. (My machine looses about 15 minutes a week, otherwise!)

VSTATS.SPACE.INTERVAL [Variable]

If the global variable VSTATS.SPACE.INTERVAL is a positive number, VSTATS displays, both graphically and alphanumerically, the utilization of Data, Atom, and VMem spaces and optionally Disk space, which is updated every VSTATS.SPACE.INTERVAL seconds.

VSTATS.SPACE.PANIC.LEVEL [Variable]

If VStats is displaying space utilization, and VSTATS.SPACE.PANIC.LEVEL is a percentage between 1 and 100 (or a fraction between 0 and 1), and any of the memory space utilizations (other than disk) exceed this percentage, VSTATS will flash its window in proportion to the excess, whether the window is occluded or not.

VSTATS.SPACE.SHOW.DISK? [Variable]

If VStats is displaying space utilization, then if VSTATS.SPACE.SHOW.DISK? is non-NIL, Disk space utilization is displayed as well, provided VStats can figure out the total disk size. If VSTATS.SPACE.SHOW.DISK? is T, the default DSK is used, for instance {DSK19} on a Dorado, or {DSK}<LispFiles> on a Dandelion. Alternate Dorado partitions or Dandelion volumes may be assigned to VSTATS.SPACE.SHOW.DISK? as well. If assigned through the options window, VStats will figure out which volumes or partitions are displayable.

VSTATS.MUTIL.INTERVAL [Variable]

If the global variable VSTATS.MUTIL.INTERVAL is a positive number, VSTATS displays, both graphically and alphanumerically, the machine utilization in terms of CPU time, time spent on disk

and Ethernet I/O, garbage collection time, and swapping time, which is updated every VSTATS.MUTIL.INTERVAL seconds.

VSTATS.MUTIL.HYSTERESIS [Variable]

If VStats is displaying machine utilization and VSTATS.MUTIL.HYSTERESIS is a positive number, the relative percentages are based on the average over VSTATS.MUTIL.HYSTERESIS intervals, otherwise they are based on the total time since VSTATS was invoked.

VSTATS.POSITION [Variable]

If the global variable VSTATS.POSITION is a POSITION, the VSTATS display will be put there, otherwise the user is prompted for a POSITION.

VSTATS.BLACK? [Variable]

If the global variable VSTATS.BLACK? is non-NIL, VSTATS displays with inverse video.

VSTATS.ALWAYS? [Variable]

If the global variable VSTATS.ALWAYS? is non-NIL, VSTATS will always update its display when its timers expire, causing its window to come to the top if it isn't already there; otherwise, VSTATS will only update the display if its window is neither partially nor wholly occluded. If it is occluded, VSTATS will, of course, continue to update its internal timers and the display will be updated the first time the timers expire after the display becomes wholly visible again.

**Defaults**

VSTATS.BLACK?	NIL
VSTATS.ALWAYS?	NIL
VSTATS.POSITION	top right corner of display
VSTATS.CLOCK.INTERVAL	1 second
VSTATS.SPACE.INTERVAL	300 seconds (5 minutes)
VSTATS.SPACE.PANIC.LEVEL	95 %
VSTATS.SPACE.SHOW.DISK?	T
VSTATS.MUTIL.INTERVAL	1 second
VSTATS.MUTIL.HYSTERESIS	20 intervals

If different values are preferred, these variables should be set by the user before loading VSTATS to affect initial display. They can of course be altered anytime using the options menu.

**Extras:**

A number of functions are required (and supplied) by VSTATS which the author believes might well be part of standard Interlisp-D, viz.,

(COVEREDWP *window*) [Function]

Returns T if *window* is partially or completely covered by some other window; NIL otherwise.

(CLOCKTICKS *interval timerunits*) [Function]

Returns the (machine dependent!) number of internal clock ticks over the interval. For instance, on the D'Lion

(CLOCKTICKS 2.5 'MINUTES) = 5211900

(ALTOPARTITIONS)

[Function]

On the Dorado, returns a list of partitions set up with an Alto exec, i.e., containing a system boot file. Especially useful with the recently added Extended VMem option, where not all partitions are bootable. Returns NIL on any other machine type. Note: this list of partitions takes between 15-20 seconds to compute.

(DISKUSEDPAGES *dsk recompute*)

[Function]

Returns the total number of disk pages in use (complementing DISKFREEPAGES). On Dorado, this is only an estimate, unless *recompute* is non-NIL in which case you wait ~ 8 seconds for the answer.

(DISKTOTALPAGES *dsk recompute*)

[Function]

Returns the total number of disk pages available (sum of DISKFREEPAGES and DISKUSEDPAGES).

---

---

**WDWHACKS**

---

---

By: Atty Mullins (Mullins.pa@Xerox.com).

Some short hacks that make window management slightly easier.

Loading this file forces the menu entries: **SLAMWS** (in the background menu) with the subitem **INSPECTORS**, and **POPSHAPE** (under "Close" in the window menu), and replaces the action for **SHAPE** (window menu) with its own call.

When **SLAMWS** is selected, you are asked for a region of the screen, and all of the windows that intersect that region are closed (by call to **CLOSEW**). If the **INSPECTORS** subitem is selected, all open (not shrunken) inspector windows are closed.

When **SHAPE** is selected, the old shape of the window is stored, and then selecting **POPSHAPE** will reshape the window to the stored shape. When **POPSHAPE** is called, the current shape is stored, so that doing **POPSHAPE** multiple times will rotate between the two shapes.

The shape/popshape hacks are useful if you want to either get something out of the way temporarily (but not lose it entirely); or to enlarge something temporarily (e.g., for doing a SEE in the typescript window).

**SLAMWS** is most useful for inspector and notecards applications, where there are a whole slew of windows that you'd like to clobber.

---



---

## WHO-LINE

---



---

By: SML (Lanning.pa@Xerox.com)

### INTRODUCTION

Need to know what package you're in? Don't know what your connected directory is? Fret not. The Who-Line is here.

The Who-Line is a window that displays this information on your screen. It is continually updated to reflect the current state of the world (thanks to an entry on BACKGROUNDFN). Additionally, items in the Who-Line can act as menu items, allowing you to change the state of the machine.

### Defining the information displayed in the Who-Line

The values displayed in the Who-Line are determined by the setting of the variable `*WHO-LINE-ENTRIES*`.

`*WHO-LINE-ENTRIES*` [Global Variable]

`*WHO-LINE-ENTRIES*` is a list that describes the items that will be displayed in the who-line. Each item in the list should be a list of up to five things: the name of the item; a form that, when evaluated, will produce the value to display; the maximum number of characters in the value; an optional function to call if the item is selected (with the mouse) in the Who-Line; an optional form that will reset any internal state of the entry when evaluated; and an optional string that describes the value displayed by the entry.

[[NOTE: Since the items on the Who-Line are evaluated rather often, it is best if they are fast and efficient (= don't CONS or allocate any space).]]

The following are standard members of `*WHO-LINE-ENTRIES*`.

`*WHO-LINE-USER-ENTRY*` [Variable]

Displays the current user in the Who-Line. Selecting this item in the Who-Line will let you change the logged in user.

`*WHO-LINE-HOST-NAME-ENTRY*` [Variable]

Displays the (ETHERHOSTNAME) of the machine you are running on.

`*WHO-LINE-PACKAGE-ENTRY*` [Variable]

Displays the package of the current TTY process in the Who-Line. Selecting this item in the Who-Line will let you switch the package of the current TTY process.

`*WHO-LINE-READTABLE-ENTRY*` [Variable]

Displays the (name of the) readtable of the current TTY process in the Who-Line. Selecting this item in the Who-Line will let you switch the readtable of the current TTY process.

**\*WHO-LINE-TTY-PROC-ENTRY\*** [Variable]

Displays the name of the current TTY process in the Who-Line. Selecting this item in the Who-Line will let you give the TTY to a different process.

**\*WHO-LINE-DIRECTORY-ENTRY\*** [Variable]

Displays the current connected directory in the Who-Line; the directory is shown in the format "Dir>Subdir>...>Subdir on {Host}". Selecting this item in the Who-Line will let you connect to another directory: the variable **\*WHO-LINE-DIRECTORIES\*** (see below) is used to produce a menu of interesting directories. If you are holding down a SHIFT key when you select an item from this menu, the directory name will be COPYINSERTed into the current tty input stream, otherwise you will be connected to that directory.

**\*WHO-LINE-VMEM-ENTRY\*** [Variable]

Displays the percentage of the VMem file that is currently being used in the Who-Line. If the VMem file is inconsistent, the number will be preceded by an asterisk ("\*"). Selecting this item in the Who-Line will let you do a (SAVEVM).

**\*WHO-LINE-SYMBOL-SPACE-ENTRY\*** [Variable]

Displays the percentage of symbol space that is currently in use.

**\*WHO-LINE-TIME-ENTRY\*** [Variable]

Displays the current time in the Who-Line. Selecting this item in the Who-Line will let you do a (SETTIME). If you hold down a shift key when you select this item, the current time will be COPYINSERTed into the current tty input stream instead.

The default value of **\*WHO-LINE-ENTRIES\*** contains all these items

#### Other ways to tailor the Who-Line

**\*WHO-LINE-ANCHOR\*** [Variable]

**\*WHO-LINE-ANCHOR\*** describes where the who-line will be displayed. If **\*WHO-LINE-ANCHOR\*** contains the symbol :TOP, the Who-Line will be anchored at the top of the screen; if it contains the symbol :BOTTOM it will be anchored at the bottom of the screen. If **\*WHO-LINE-ANCHOR\*** contains the symbol :LEFT, it will be anchored to the left side of the display; if it contains the symbol :CENTER it will be centered on the screen; if it contains the symbol :JUSTIFY it will run the width of the screen; if it contains the symbol :RIGHT it will be anchored to the right side of the screen. Finally, if **\*WHO-LINE-ANCHOR\*** is a POSITION, it will be used as the lower left corner of the Who-Line. The default value is (:CENTER :BOTTOM).

**\*WHO-LINE-NAME-FONT\*** [Variable]

The font used to display the names of the items in the who-line. The default is HELVETICA 8 BOLD.

**\*WHO-LINE-VALUE-FONT\*** [Variable]

The font used to display the values in the who-line. The default is GACHA 8.

**\*WHO-LINE-COLOR\*** [Variable]

The color of the Who-Line. Legal values are the keywords :WHITE and :BLACK. The default is :WHITE.

**\*WHO-LINE-BORDER\*** [Variable]

The border width of the Who-Line window. The default is 2.

**\*WHO-LINE-TITLE\*** [Variable]

The title of the Who-Line window. The default is NIL.

**\*WHO-LINE-DISPLAY-NAMES?\*** [Variable]

If **\*WHO-LINE-DISPLAY-NAMES?\*** is true, the names of items in the who-line will be displayed; otherwise they will not be shown. The default value is T.

**\*WHO-LINE-UPDATE-INTERVAL\*** [Variable]

The number of milliseconds between updates of the who-line. The default is 100 milliseconds.

### Installing new Who-Line options

Changing the above variables has no direct effect on the who-line. These values need to be installed in the Who-Line before they can take effect.

**(INSTALL-WHO-LINE-OPTIONS)** [Function]

INSTALL-WHO-LINE-OPTIONS installs the above options in the Who-Line, and updates the Who-Line accordingly.

The Who-Line supports an easy way to interactively add or remove entries. If you click on the Who-Line while holding down the EDIT or CONTROL key, you will be given a chance to add or remove items from the Who-Line.

**\*WHO-LINE-ENTRY-REGISTRY\*** [Global Variable]

A list of all known Who-Line entries. This is used to construct the menu of possible new entries for the Who-Line.

### Who-Line process state

The who-line entry **\*WHO-LINE-TTY-STATE-ENTRY\*** tries to display the current state of the TTY process.

**\*WHO-LINE-TTY-STATE-ENTRY\*** [Variable]

A Who-Line entry that displays the "state" of the current TTY process in the Who-Line. The typical state of a process is the name of the function that is currently running in that process. This simple minded result can be altered by use of the following items.

[[NOTE: Because of the nature of the Lisp scheduler, this information is almost always out of date.]]

The Who-Line "state" can be explicitly controlled from code. If the special variable `*WHO-LINE-STATE*` is bound, its value is taken to be the state of that process. You can use this feature to provide visual indication of the state of your code by using the programming idiom:

```
(LET ((*WHO-LINE-STATE* indicator))
  (BLOCK)                ;Give the Who-line a chance to run
  ...your-code...)
```

This will run the `...your-code...` with the Who-Line state of the process set to (the value of) `indicator`. The call to `BLOCK` insures that the Who-Line has a chance to update before `...your-code...` is run.

`*WHO-LINE-STATE-UNINTERESTING-FNS*` [Global Variable]

If there is no declared who-line state (via a `WITH-WHO-LINE-STATE` form), then the name of the function that is currently running is used as the who-line state. However, if the function is on the list `*WHO-LINE-STATE-UNINTERESTING-FNS*`, the function that called *it* is used instead. The default value of `*WHO-LINE-STATE-UNINTERESTING-FNS*` is `(BLOCK AWAIT.EVENT)`.

`WHO-LINE-STATE` [Property]

If the function that is currently running has a `WHO-LINE-STATE` property, the value of that property is used as the who-line state. This is used to convert functions like `\TTYBACKGROUND` to meaningful values like "TTY wait".

`(WHO-LINE-REDISPLAY-INTERRUPT)` [Function]

Updates the Who-Line. It is intended that this function be installed on an interrupt character, so that the user can easily force an update of the Who-Line. For example,

```
(ADVISE 'CONTROL-T 'BEFORE '(WHO-LINE-REDISPLAY-INTERRUPT))
```

will cause a `↑T` interrupt to update the Who-Line as well as its current behavior of printing state information in the Prompt window. Alternatly, you can define a new interrupt character that will force an update of the Who-Line;

```
(INTERRUPTCHAR (CHARCODE ↑U) '(WHO-LINE-REDISPLAY-INTERRUPT) 'MOUSE)
```

will cause the Who-Line to be updated whenever the user hits a `↑U`.

### Other interesting things

`*WHO-LINE-DIRECTORIES*` [Global Variable]

A list of interesting directories used to generate a pop-up menu of directories to connect to when you select the `DIRECTORY` item in the Who-Line. The default value is a list containing just your `LOGINHOST/DIR`. When the Who-Line notices that you have changed your connected directory, it updates this list to contain the new directory.

`(CURRENT-TTY-PACKAGE)` [Function]

Returns the name of the package of the current TTY process. This function is used in the default value of `*WHO-LINE-ENTRIES*`.

`(CURRENT-TTY-READTABLE-NAME)` [Function]

Returns the name of the readtable of the current TTY process, or the string "Unknown" if it can't figure out the name. This function is used in the default value of `*WHO-LINE-ENTRIES*`.



**(SET-PACKAGE-INTERACTIVELY)****[Function]**

Pops up a menu of currently defined packages. If the user selects one of them, the current package is changed to the selected package.

**(SET-READTABLE-INTERACTIVELY)****[Function]**

Pops up a menu of currently known readtables. If the user selects one of them, the current readtable is changed to the selected readtable.

---

---

## WHOCALLS

---

---

By: Bill van Melle (vanMelle.pa@Xerox.com)

This file contains two useful functions for quick crossreference:

(whocalls *callee usage*) [Function]

maps over all symbols in the current environment, looking for any function that mentions callee according to usage:

values for usage:

USES VAR VARS BOUND USEDFREE GLOBALS

All mean: mention as a variable

NIL CALLS

means calls as a function

(distribute.callinfo) [Function]

inverts all of the call, use global, use free, bound relations for functions, variables from compiled code. Operates by mapping over all symbols in the sysout that are defined as compiled code, and analyzing their definitions. Anything that is called has a CALLEDBY property of all of the things that call it; any variable bound has a BOUNDBY with the list of functions that bind it, variables that are used globally have a USEDGLOBALBY and variables that are used freely have a USEDFREEBY.

(References from interpreted code, etc are not detected, so it isn't 100% guaranteed that if something doesn't have a CALLEDBY that it isn't called.....)

---



---

## XCL-BRIDGE

---



---

By: Jan Pedersen (pedersen.PA @ Xerox.com)

XCL-BRIDGE is a module that assists in the transformation of ascii Common Lisp source files to Lisp managed files and vice versa. In the text-to-managed-file direction, user interaction is employed to repair read-in forms before establishing a resident image of a Lisp managed file. In the managed-to-text-file direction, a few simple transforms are employed to translate common filepackagecoms to equivalent Common Lisp forms.

All entry points are external to the "XCL" package.

(XCL:TEXT-TO-MANAGED-FILE *pathname filename &key (package "USER")*  
*(readtable "XCL") (read-base 10) (combine-comments t)*) [Function]

Reads an ascii lisp source file named by "pathname" and converts it to a managed file with rootname "filename". The package, readtable, and read-base employed to read the ascii file may be specified via keyword arguments, or defaulted, as shown. If the reader environment arguments are defaulted and the source file has a emacs-style "mode line", then the package and read-base will be as indicated by the "mode line". The "combine-comments" keyword controls whether adjacent comments at the same ";" level should be combined when generating sedit-style comments for the converted file.

Note that forms are only read from the ascii lisp source file, not evaluated. It is assumed that the converted file should be made (via "il:makefile") and compiled before any evaluation should be attempted.

Text-to-managed-file proceeds incrementally and interactively to convert the specified file. First all the forms are read, and presented to the user for editing (via Sedit). If the user accepts this primary phase, a filecoms is generated and again, presented to the user for editing. If the user accepts the generated filecoms, a file (and its contained definitions) is instantiated, completing the conversion.

(XCL:MANAGED-TO-TEXT-FILE *filename pathname &key (package "USER")*  
*(readtable "XCL") (print-base 10)*) [Function]

Prints a managed file, with rootname "filename", whose source definitions must be resident, to an ascii file "pathname" in a form suitable for reading by any Common Lisp reader. The read-print environment of the managed file may be overwritten via the keyword arguments "package", "readtable", and "print-base". Many Interlisp "filepackagecoms" are translated to their Common Lisp equivalents. For example, "il:declare:" forms are transformed to "eval-when" forms and "il:files" forms are transformed to "require" forms. As an additional convenience, defdefiners are printed as equivalent defmacros.