Before you can print the Medley LispUsers' tedit files you must do the following:

1. If you aren't running in Lispcore you need to load {eris}<tedit>tedithcpy.locm
2. In IL:\ASCIITONS change entry 183 so that it reads 183 instead of what it normally reads.

Doing these two things causes the Envos logo to print correctly. To print some of the LispUsers' modules you must have some functions loaded. The following table shows what you must have loaded to print certain LispUsers' modules:

Module name	File needed to print or display correctly
equation examples	Mathtons.lcom equations.lcom and set il:|EQ.UseNSChars| to T
Doc-objects	docobjects.lcom (loads dateformat.lcom)
hpgl	sketch.lcom

LISPUSERS' MODULES MANUAL

en·vōs

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

TABLE OF CONTENTS

[This page intentionally left blank]

{ERIS}<LISPUSERS>LYRIC>LISPUSERS-DEPENDENCIES.TEDIT
Edited 12/16/86 Susana Wessling

This is a list of the LispUsers files that are to be released and their dependencies. Please add new LispUsers files to this list when you write them onto <lispusers>lyric>. If any of the packages on this list have changed or are not to be released, please remove them.
Key:
* internal release only
+ release source AND dcom files
indented: dependent on the above package

```
access.;1
ACE.;1
        ACE-APPLEDEMO.;1
        ACE-BOUNCINGBALL.;1
        ACE-FOUETTE.;1
ACTIVEREGIONS.;1
ACTIVEREGIONS2.;1
AIREGIONS.;1
        AIREGIONS-DEMO.;1
animate.;1
*ARCHIVETOOL.;1
ARITHDECLS.;1
ARRAYSORTER.;1
AUTOSAMEDIR.;1
AUXMENU.;2
BACKGROUND.;1
        +BACKGROUNDDEMO.;1
BACKGROUNDIMAGES.;3
        +BACKGROUND-*.BITMAP
        +BACKGROUND-*.PRESS
BackgroundMenu.;1
BICLOCK.;1
BIZGRAFIX.;1
bltdemo.;1
BMFROMW.;1
BOUNCE.;1
BOUNDARY.;1
BOYERMOORE.;1
        BOYERMOOREDATA.;1
BQUOTE.;1
        BQUOTEGACHA10-C0.DISPLAYFONT
        BQUOTEGACHA12-C0.DISPLAYFONT
        BQUOTEGACHA8-C0.DISPLAYFONT
BTMP.;1
        BTMP-DEBUG.;1
BUGREPORT.;1
CALENDAR.;2
        PROMPTREMINDERS.;1
CCACHE.;1
CD.;1
CD-COMMAND.;1
CHANGEPRINTER.;1
+CIAPROPOS.;1
COLORNNGS.;1
COMHACK.;1
COMPAREDIRECTORIES.;1
COMPARESOURCES.;1
COMPARETEXT.;1
COMPILEFORMSLIST.;1
CONNTITLE.;1
COURIERDEFS.;1
COURIEREVALSERVE.;1
COURIERIMAGESTREAM.;1
COURIERSERVE.;1
        REMOTEGRAPHER.;1
        REMOTEPSW.;1
CROCK.;1
*CRYPT.;1
DATEFNS.;1
DEDITAUG.;1
DEDITICON.;1 (no .dcom)
DEDITK.;1
DEFAULTICON.;1
DEFAULTSUBITEMFN.;1
DINFOEDIT.;1
DIRECTORYTOOLS.;1
DIRGRAPHER.;1
DIRMENU.;1
```

```
DLIONFNKEYS.;1
DOCTOR.;1
DONZ.;1
*DRAWFILE.;1
DSL.;1
dumper.;1
+DUMPLOAD.;1
EDITBG.;1
EDITFONT.;1
+EDITKEYS.;1
EDITRECALL.;1
EMACS.;1
EMACSUSER.;1
EQUATIONFORMS.;1
EQUATIONS.;1
EXEC.;1
+EXECFNS.;1
FACEINVADER.;1
FASTBITMAPBIT.;1
*FILECACHEMSGWINDOW.;1
+FILEOBJ.;1
FILEPERCENT.;1
FILLPRINT.;1
FILLREGION.;1
FINGER.;1
FLAGBROWSER.;1
FLOPPY4.;3
FONTMENU.;1
FULLSCREEN.;1
GKS.;1
GKSEXTERN.;1
GKSINTERN.;1
GKSMATRIX.;1
GLISPA.;1
        GLISPB.;1
        GLISPDWINDOW.;1
        GLISPGEV.;1
        GLISPGEVAUX.;1
        GLISPGEVTYPE.;1
        GLISPR.;1
        GLISPTEST.;1
        GLISPVECTOR.;1
graphcalls.;1
*GREP.;1
HANOI.;1
HASHBUFFER.;1
HASHDATUM.;1
HEADLINE.;1
HISTMENU.;1
IDEASKETCH.;1
IDLEHAX.;1
IMAGEWRITER.;1
*IMTEDIT.;1
        IMNAME.;1
        IMTOOLS.;1
        IMTRAN.;1
INSPECTCODE-TEDIT.;1
IRISCONSTANTS
        IRISDEMOFNS
        IRISIO
        IRISLIB
        IRISNET
        IRISSTREAM
        IRISVIEW
        LOADIRIS
        SFFONT
        GACHAE.LC1-SF;2
        GACHAE.LC2-SF;2
        GACHAE.NUM-SF;2
        GACHAE.S1-SF;2
        GACHAE.S2-SF;2
        GACHAE.UC1-SF;2
        GACHAE.UC2-SF;2
JARGON.;1
        JARGON.DB
KAL.;1
KEYOBJ.;1
KINETIC.;1
*LAFITEHIGHLIGHT.;1
LCROCK.;1
```

```
LIFE.;1
LISTEN.;1
LoadPatches.;1
LOGOCLOCK.;1
LSET.;1
MACWINDOW.;1
MAGNIFIER.;1
magnifyw.;1
*MAILOMAT.;1
MAKEGRAPH.;1
MANAGER.;1
MATHFNS.;1
*MESATOLISP.;1
MOVE-WINDOWS.;1
multimenu.;1
MULTIW.;1
MUSICKEYBOARD.;2
NOTEPAD.;1
        NOTEPAD-CORESTYLES.;1
NQUEENS.;1
PACMAN.;1
PAGEHOLD.;1
PARSER.;1
PARSERG.;1
PATCHUP.;1
PCDAC.;1
PEANO.;1
PERFORMTRAN.;1
PIECE-MENUS.;1
PLAY.;1
PLOT.;2
        PLOTEXAMPLES.;1
        PLOTOBJECTS.;1
PLURAL.;1
+PQUOTE.;1
PREEMPTIVE.;1
PRESSFROMNS.;1
PRESSTOIP.;1
        DPRESS.;1
PRINTERMENU.;1
Proofreader.;1
        ANALYZER.;1
        SpellingArray.;1
PULLDOWNMENUS.;1
+QIX.;1
*READBRUSH.;1
RECORDPRINT.;1
REGION.;1
RESETMACROS.;1
ROTATEBM.;1
*RPC.;1
        RPC-EXAMPLE.;1
        RPC-EXAMPLECLIENT.;1
        RPC-EXAMPLESERVER.;1
        RPC-EXAMPLEUSER.;1
        RPC-LUPINE.;1
        RPCEVAL.;1
        RPCEVALCLIENT.;1
        RPCEVALSERVER.;1
*SAMPLER.;1
SERVERSTATUS.;1
SETDEFAULTPRINTER.;1
SETF.;1
SHOW.;1
SIGNAL.;1
SINGLEFILEINDEX.;1
SLIDEPROJECTOR.;1
SNAPSCROLL.;1
SOLITAIRE.;1
SPACEWINDOW.;1
sprint.;1
STARBG.;1
STOCKICONS.;1
STYLESHEET.;1
+SUPERMENUEDIT.;1
+SUPERMENUS.;1
SYSTAT.;1
+TEDITKEY.;1
THERMOMETER.;1
        THERMOMETERDEMO.;1
```

```
TILEDEDIT.;1
TIMEPANEL.;1
+TINYTIDY.;1
TMENU.;1
TOGMENU.;1
TRACEIN.;1
TRAJECTORY-FOLLOWER.;1
        TRANSOR.;1
TRUEHAX.;1
TSET.;1
TTY.;1
ttyio.;1
TURING.;1
TWOD.;1
TWODGRAPHICS.;1
UNBOXEDOPS.;1
utilisoprs.;1
VMEMSTATE.;1
VSTATS.;1
+WAM.;1
WDWHACKS.;1
WINK.;1
WINNER.;1
WORM.;1
+YAPFF.;1
```

Other TEDIT files: [on {eris}<lisp>koto>lispusers> or {eris}<lispusers>koto>]

```
Release-intro.tedit           intro for released doc's
Release-rules.tedit           rules for released doc's
Internal-intro.tedit          intro for internal doc's
Internal-rules.tedit          rules for internal doc's
blankpage.tedit                       blank page for printing screwups...
documentationtemplate.tedit   long template for doc's
easytemplate.tedit            short template for doc's
lispusers-rules.tedit  old LU rules
RELEASE-INFO.tedit            This document
LISPUSERS.TEDIT                       Short package summaries
```

Subject: Re: Lispusers packages (updates)
In-reply-to: bloomberg.pa's message of 16 Dec 86 12:38 PST
To: bloomberg.pa
cc: LispUsers↑.x
Reply-to: Wessling.pa
Format: TEdit

I received this inquiry today:

Date: 16 Dec 86 12:38 PST
From: bloomberg.pa
Subject: Lispusers packages (updates)
To: wessling.pa
cc: bloomberg.pa

When I replace text or documentation of a Lispusers package with new versions, should I message you so that you can do the appropriate thing(s) vis-a-vis phylum?

--Dan

...and I thought this would be a good time to just give an update on how to handle the lispusers packages.

--- Directories:

{eris}<lisp>koto>lispusers> is where the frozen, released packages for Koto were put.
{eris}<lispusers>koto> is where the Koto packages that have been changed since release should be put.
{eris}<lispusers>lispcore> is where new packages that run in LispCore and updates of old packages that are updated to Lispcore should be kept.

--- Creating new packages:

if you create a new package that runs in LispCore, i.e. to be release with Lyric, write it onto {eris}<lispusers>lispcore>.

--- Editing old packages:

if you want to change a package that was released with Koto, make changes to the latest version (if it hasn't changed since release this is on {eris}<lisp>koto>lispusers> or if it has been updated, it should have been put on {eris}<lispusers>koto>) and copy it to:
{eris}<lispusers>koto> if it **only** still runs in Koto
{eris}<lispusers>Lispcore> if you **know** it runs in Lyric.

--- What should be included:

No package will be released from LispUsers without:
- Source file
- Compiled file

- Documentation

Documentation can be made through the template ({eris}<lispusers>LispCore>easytemplate.tedit) or by using the new lispusers package, DOCUMENT (on {eris}<lispusers>koto>).

--- for more information:

{eris}<lispusers>lispcore>LispUsers-rules.tedit, LispUsers-Summary.tedit, LispUsers-dependencies.tedit. I will store this on {eris}<lispusers>lispcore>lispusers-info-msg.tedit.

--- Nice things to do:

When you create a new package, please:
- send a note to LispUsers↑ announcing it.
- add it to LispUsers-summary.tedit (a list of all the lispusers packages)
- add it to LispUsers-dependencies.tedit, which is a list of all the lispusers packages and which files they need to run.
- Edit LispUsers-dependencies.tedit to reflect the current state of Lispcore lispusers files (right now it is just the info from Koto).

If you created a package for Koto, please try to update it to Lispcore and store it onto <lispusers>lispcore>. If the package is un-updatable or useless in LispCore, please take it off of the aforementioned lists and/or message me about it.

--- and to answer the question that started all this:

Just put your packages on the directories mentioned. Any copying will be done.

Thank you very much for your cooperation....

Susana...

Date: 23 Jan 87 17:21 PST
From: Wessling.pa
Subject: {eris}<lispusers>lyric> exists!
To: LispUsers↑.x
Really-from: The Tired Floppy Lady (let's all go home... it's Friday...)
Reply-to: Wessling.pa

NOTICE: If you are either a user or a creator/maintainer or ANY LispUsers module, please please please do not delete this message. I am sick of resending information...

All files that existed on {eris}<lispusers>lispcore> as of NOW have been moved to {eris}<LispUsers>Lyric>.

From now on, follow these rules for storing new or changed LispUsers modules:

- If the module was created or changed in Koto, please store it under {eris}<lispusers>koto>.

- If the module was created or changed in the {eris}<lyric>basics> sysout, please store it on {eris}<lispusers>lyric>.

- If the module was created or changed in the {eris}<lispcore>next> sysout, please store it on {eris}<lispusers>lispcore>.

Though this may seem a bit complicated, it will help us to keep track of which modules are assured of working where.

****
RULES:

AND NOW... I am once again going to remind you of how you can make our lives a bit easier when interacting with LispUsers:

All modules that are stored on ANY of the aforementioned directories MUST have:

- source file
- compiled file (.LCOM or .DFASL for Lyric, .DCOM for Koto)
- documentation

If you do not store any of the above with your module, it WILL NOT be released.

In writing documentation, you can use EasyTemplate.tedit (stored on any/all LU directories) to make life easier. For a long version, see DocumentationTemplate.tedit. Please please please note in your documentation:

- any dependencies between modules
- if the module is for internal use only

For further and more detailed rules, see Lispusers-rules.tedit.

If you are nice, I would appreciate your updating all of the <lispusers>lispusers* files. These files contain information about releasable modules that is very important. You should look into these files if
- your module requires other modules to be loaded with it.
- your module is for internal release only
- your module is new

****
QUESTIONS/REMINDERS:

It has been brought to my attention that there are some modules that are stored on {eris}<lispusers>lyric> that are definitely NOT lyric compatible. Please do not do this. It will not make anyone's job easier. Such modules should go on <lispusers>koto>. Please move them if you are responsible for such horrific actions!

If you are responsible for or just have a vested interest in some module that is in <lispusers>koto> but has not been updated, please see to getting it updated and put into <lyric> or <lispcore>. If it is not done, it will not go out with Lyric.

For those who are anxiously waiting for an NS LispUsers directory... I'm working on it. I will send out further information if it happens.

****
THAT'S ALL FOLKS...

Sorry for the long message, but these things just had to be said....

Susana

—End of message—

LISP USERS' RULES

This document describes the rules and procedures for Xerox Lisp "Lisp Users'" modules. This document is mainly for Lisp Users' module writers, but users should also understand the rules. The deadline for Medley Lisp Users' module submissions is August 5, 1988.

DEVELOPING A LISP USERS' MODULE

A Lisp Users' module is a useful program made available to the general Xerox Lisp programming community. Neither the author nor the custodian of the Lisp Users' directory imputes any warranty of suitability or responsibility for errors.

Lisp Users' modules should be easily distinguishable from released library modules. In particular, this means that a Lisp Users' module may not have the same name as a Library module and should be visibly different. Lisp Users' modules derived from released software should be announced to the public only after communicating with the organization responsible for that released software.

Testing is important. If you make significant changes to a Lisp Users' module, enlist developers at your site as alpha testers. Avoid having to rerelease a package within hours of its announcement because of fatal bugs. A Lisp Users' module is not shoddy software; it is software made available outside the regular release channels.

Try not to flood your user community with a constant flow of new versions (or messages) so your messages can be appreciated rather than discarded without reading. If your module is undergoing continual changes, adopt a release strategy of regularly spaced, well tested releases. Your users will thank you.

LISP USERS' MODULE OWNERSHIP

A module submitted for Lisp Users' remains the "property" of the submitter. Others may not make changes, except for their own private use, without negotiating with the owner (who may already be making similar or incompatible changes).

As the owner of a module, you are not required to fix bugs, but if not, you must be willing to transfer ownership (permanently) to someone who volunteers to fix them. Ownership may pass back and forth among several people as long as they agree.

Like all software developed at Xerox, Lisp Users' modules are officially the property of Xerox. You should run with the COPYRIGHT option set to produce an appropriate Xerox copyright in the source.

A Lisp Users' module may become so useful that it becomes part of a standard Xerox Lisp release and is thereafter supported. Ownership then passes to the Xerox product organization.

SUBMITTING LISP USERS' MODULES

If you are not an internal user, you should submit your new module to Xerox either through e-mail or on a floppy. External users should make sure that they include all relevant information, such as documentation containing an e-mail or US mail address where he/she can be reached. External users are also held responsible for the support of their modules.

SUBMITTING FILES TO LISP USERS'

As with released software, it is important to submit not just the resulting product, but all the files needed to build and maintain a Lisp Users' module:

1. the runnable compiled file (.LCOM or .dfasl)
2. documentation describing it, following the set formatting rules (see below)
3. a source file that can be released (optional)
4. data files, if needed

Packages submitted once are released once. Do not assume that a package submitted in one release will be automatically released in subsequent releases.

DOCUMENTATION

Documentation, essential to any module being used effectively, should be put on the submitted floppy. No modules will be released without documentation. Documentation can be as simple as a paragraph describing what the module does and how to use it, or it can be as extensive as a dozen-page user manual. All modules should have a file with a .TEDIT extension. Formatting should be done according to the rules outlined in the appropriate (standalone or networked user) Lisp Users' Template. All users, external users included, should follow the Lisp Users' Template rules. If the documentation is large and formatting time consuming, you can also produce an interpress file (with the .ip extension), as well as submitting a .TEdit file. (Be sure to update the interpress file if you update the documentation!) Documentation should include the full electronic mail address of the submitter.

COMPATIBILITY WITH LISP USERS'

Any submitted Lisp Users' files should be compilable in a "vanilla" environment. The file itself should load in any auxiliary modules under a suitable (DECLARE: EVAL@COMPILE --) when necessary.

Thanks for your cooperation.

Lispusers Summary

A one-line description of (most) LispUsers packages.

All files listed here should be in {eris}<lispusers>lispcore>. If you want to look at the list for Koto files, see <lispusers>koto>, the same file. Please add any new files you create to this list.

---NO TEDIT FILE means that no documentation is needed.
---UNSUPPORTED means that there is no author or the author is unreachable.
---an asterisk before a file means its part of the previous package.
---INTERNAL means that it is not for release outside of Xerox
---SOURCE means that you load the source & not the DCOM
---Two dashes (--) before it means that it exists in Koto but hasn't been updated for lyric yet.

Module	Comments
--ACCESS	
--Ace	Animation Compiler; includes graphics editor.
--* ACE-EDIT, ACE-Main, ACE-Prim, ACE-AppleDemo (Source), Ace-BouncingBall (Source), ACE-Fouette (Source)	
--ActiveRegions	Mouse sensitive regions in a window.
--ActiveRegions2	Rewrite of ActiveRegions
--AIRegions	
--* AIREGIONS-DEMO (Source)	
--Analyzer	Part of proofreader
--Animate	Smoothly move arrow, finger around on screen.
--ArithDecls	For use with decl
--ArchiveTool	(INTERNAL) Cedar archive system interface
--ArraySorter	
--AutoSameDir	Put sources back where you got them at MAKEFILE
--Auxmenu	Useful middle-button menu in background
--Background	BITBLT to screen background or obscured windows.
--* BACKGROUNDDEMO	
--BackgroundImages	screen backgrounds, including rhine, two dollar bill, castle
BackgroundMenu	Useful menu in the background
BiClock	Swedish clock
BITMAPFNS	
--BizGrafix	Pie & bar chart, line graph creation
--BLTDemo	graphics demo/idle hack
--BMFromW	bitmap from window
--Bounce	Idle hack with lines
--Boundary	
BoyerMoore	The Boyer-Moore Theorem Prover
--* BoyerMooreData (Source)	
--BQuote	Yet another backquote macro (like Common Lisp's)
BSEARCH	
--BTMP	Basic Text Macro Processor
--* BTMP-Debug	
--BugReport	
--Calendar	calendar/appointment-reminder program
--CCache	keep files on local disk, like FILECACHE but less automatic
CD	Connect-to-directory command
--CD-Command	Connect-to-directory command
--Changeprinter	Interface for dealing with printers
CHATSERVER	
CHECKSET	
--CIApropos	Case Independent apropos
CIRCLPRINT	
--ColorNNGS	For use with busmaster and Number 9 Graphics Card on 1108
--ComHack	Comments in if and fors
COMMENTSTRINGS	
COMMON-MAKEFILE	Creates plain-text exportable Common Lisp source code files
COMMWINDOW	
--CompareDirectories	What files are different on two directories
CompareSources	Compare two lisp files
--CompareText	Compare two text files
COMPILEBANG	
--CompileFormsList	
--ConnTitle	Show where I'm connected
--CourierDefs	Implementation of XNS protocol for remote procedure call
--CourierEvalServe	EVAL server using courier
--CourierImageStream	Image stream server using courier
CourierServe	courier server
--Crock	Analog clock
--Crypt	(INTERNAL)
--Datefns	
--DEditAug	
--DEditHardcopy	(NEW) hardcopy of Dedit window prettyprints whole function

--DEditIcon	An icon for Dedit
DEditK	New DEdit menu with common combo's
--DefaultIcon	new default icon for shrunk windows
--DefaultSubitemFN	
DES	
DIGEST	
DInfoEdit	
--DirectoryTools	
--Dirgrapher	Shows a graph of directory structure
DLionFNKeys	Dandlion keys (center/bold/etc) for 1132 users
DMSG	
--Doctor	Infamous Eliza program (computer shrink)
--Donz	talking windows
--DPress	(INTERNAL) take apart press files
DSKTEST	
--DSL	Digital Speech Lab, uses Busmaster speech analysis
--Dumper	(INTERNAL) Alto Exec DUMP files
--DumpLoad	Alto Exec DUMP files
--EditBG	Edit background & background border shade
--EDITFONT	Create & edit display fonts
--EditKeys	Dandlion keys (center/bold/etc) for 1132 users
--EditRecall	
EMACS	EMACS commands on top of tedit
--EMACSUser	Use EMACS as your programming environment
--EquationForms	TEdit file=EquationEditor.tedit; for editing equations
--Equations	TEdit file=EquationProgram; programmatic interface
--Exec	Extra exec windows (like Listen)
--ExecFNS	INTERNAL
--FaceInvader	A game
--FastBitmapBit	
FILECACHE	
FILECACHE-BROWSER	
FILECACHE-DECLS	
FILECACHE-HOSTUP	
FILECACHE-SCAVENGE	
FILECACHEMSGWINDOW	
--FileObj	Files as image objects
--FilePercent	Lanning
FILEWATCH	
--FillPrint	
--FillRegion	
--Finger	Who else is running Finger on the net?
--FlagBrowser	
--FontMenu	
--FullScreen	
--GKS	Graphics Kernel Standard implementation
--* GKSExtern	
--* GKSIntern	
--* GKSMatrix	
--GLISP	Compiler for GLisp, an object-oriened Lisp language
--* GLISPA	
--* GLISPB	
--* GLISPDWINDOW	
--* GLISPGEV	
--* GLISPGEVAUX	
--* GLISPGEVTYPE	
--* GLISPR	
--* GLISPTEST.LSP	
--* GLISPVECTOR.LSP	
--* GLISP.tty	
--GraphCalls	Graph calls from interpreted,compiled code
--GREP	NEW. Search file(s) for strings, e.g. phone book.
--Hanoi	Graphics demo/idle hack
--HashBuffer	
--HashDatum	
--Headline	Big titles on the screen
--HistMenu	History list as a menu
--IdeaSketch	
IdleHax	collection of idle hacks
--IdleSwap	(NEW) idle hack
--ImageWriter	output to Apple image writer
--IMName	(INTERNAL) edit Interlisp Manual
--IMTedit	(INTERNAL) convert Interlisp Manual to TEdit
--IMTools	(INTERNAL) Tools for dealing with Interlisp Manual
--IMTran	(INTERNAL) Translate Interlisp Manual
--InspectCode-TEdit	
INVISIBLEWINDOW	
--Jargon	definitions from the hacker's dictionary
--Kal	b/w or color kaleidoscope (also in Idlehax)
--? KeyboardTool	

Keyobj	key image-objects
--Kinetic	Graphics demo/idle hack
--LCrock	Clock in the logo window
Life	Conway’s game of life, as an Idle hack
LISPXCONVERT	
--Listen	Lisp Executives from the background menu
--LoadPatches	
--LogoClock	Another clock in the logo window
--LSet	Lists as sets
MacWindow	Shrink & expand windows like a MAC
--Magnifier	Magnify areas of the screen
--MagnifyW	
--MailoMat	INTERNAL
--MakeGraph	help for Grapher users making graphis
Manager	Window/menu file package interface
MANDELBROT	
--MathFNs	trig, complex functions, constants
MERGE-FILEGEN	
--Move-Windows	
--MTP	(INTERNAL) Mail Transfer Protocol for Lafite<->Tops-20
--MultiMenu	Attached menus in groups
--MultiW	Heirarchical window environment
--MusicKeyboard	
--Notepad	Graphics paint program
--* NOTEPAD-CORESTYLES (SOURCE)	
NOVAFONT	
NSCHATSERVER	
NSDISPLAYSIZES	
NSMAINTAIN	
NSREADERPATCH	
--NSthasize	(INTERNAL NEW) Convert GV Distribution list to NS
--NQueens	Graphics demo
--Pacman	Game
--Pac-Man-Idle	(NEW) Idle hack
PageHold	Changes "window full" behavior on scrollers
--Parser	Parser generator for making new parsers
--Patchup	
--PCDAC	A-to-D and D-to-A using BusMaster and PC boards
--Peano	Graphics demo
--Performtran	add clisp word to record package
--Piece-Menus	
PLANETS	
--Play	Tunes on 1108/1186 beeper
--Plot	Making plots
--* PlotExamples	Some samples
--* PlotObjects	Plots as image objects in documents
--Plural	Plural of words
--PQuote	Prettyprint (QUOTE x) as ’x.
--Preemptive	make scheduler preemptive (caveats)
--PressFromNS	
--PressToIP	
--PrinterMenu	
--PromptReminders	reminders at a given time, used by Calendar
ProofReader	Spelling checker in Tedit
PUPCHATSERVER	
--* SpellingArray	
--PullDownMenus	
--QIX	Shrager
--RecordPrint	
--Region	
--RemoteGrapher	Grapher over XNS connection on another machine
--RemotePSW	Someone elses Process Status WIndow on your screen
--ResetMacros	
--RotateBM	Rotate bitmaps
--RPC	(Internal) Cedar-style PUP based Remote Procedure Call
--* RPC-Example	
--* RPC-ExampleClient	
--* RPC-ExampleUser	
--* RPC-Lupine	
--* RPCEVAL	
--* RPCEVALCLIENT	
--* RPCEVALSERVER	
RS232CHATSERVER	
--Sampler	Graphics demo
SCREENPAPER	
--ServerStatus	
--SetDefaultPrinter	
--SetF	Common Lisp style SETF
--Show	

--Signal	Mesa-style signals
SingleFileIndex	Add index to ListFiles output
--SlideProjector	Cycle thru tedit file of "slides"
--SnapScroll	Scrollable "snap" windows
--Solitaire	Graphics demo/Idle hack
--SpaceWindow	space allocation use in a window
--SPrint	
--StarBG	Stars in the background/Idle hack
--? StockIcons	
--StyleSheet	create block of menus
--SuperMenus	
--* SuperMenuEdit	
--Systat	control-T puts up window w/graphic display
TCPCHATSERVER	
TEditKey	New TEdit commands as various meta-characters
--Thermometer	
--* THERMOMETERDEMO	
--TileDEdit	Dedit windows place themselves so they don't overlap
--TimePanel	1108 Maintenance panel => clock
TinyTidy	Move icons over to edge of screen
--TMenu	menus that stuff input buffer, pull down menus
--TogMenu	
--TraceIn	stepper/tracer for debugging
--Trajectory-Follower	animation of following a trajectory
--Transor	
--* TSet	
--TrueHax	
--TTY	
--TTYIO	
--Turing	Turing machine simulator
--TwoD	Two dimensional graphics package
--TwoDGraphics	Two dimensional graphics package
UnboxedOps	Unboxed arithmetic for dandetiger's
UPCSTATS	
--UtilISOpr	Additional iterative operators
--VMemState	Turn on/off VMEM.PURE.STATE
--VStats	Show time, space used
--WAM	Window Action Menu
--WDWHacks	
WHO-LINE	
WHOCALLS	
--Wink	Graphics demo
--Winner	
--Worm	Idle hack
--Yapff	Yet Another Page Full Function

>>MODULE NAME<<

By: >>Your Name<< (>>Your net address<<)

Uses: >>Other modules necessary to run this one<<

>>Type INTERNAL here if the file is for Internal Use Only<<

This document last edited on >>DATE<<.

INTRODUCTION

>>This paragraph should be replaced by an overview of your module. The attached Lisp Users' Template explains the documentation conventions to be used for each Lisp Users' modules.<<

MODULE EXPLANATIONS

>>Functions, Variables, and Lisp Code Examples<<

It is usual to first give the name of a function, then describe its purpose and each of its arguments. When the name of a function is first given, it is set off like this:

(IMAGEFNSCREATE *DISPLAYFN IMAGEBOXFN PUTFN GETFN COPYFN*) [Function]

The function name is in 10-point regular Modern, all caps. Arguments are in 10-point italic Modern, in all caps, mixed case, or lowercase, as they appear in the system. Variables look like functions, except that the word ‘‘Variable,’’ enclosed in **square brackets**, follows the variable name. Please note that these are the characters [], not the parentheses

```
This is an example of code. It is in 10-point Terminal font.
```

Function names, commands, file names, and the like are in 10-point modern.

Be sure to include the following information in any module explanations:

• any file dependencies

• definitions of all arguments

• module, function variable, etc. limitations

• a liberal number of examples for all functions, variables, etc.

LISP USERS' TEMPLATE

Updated by: Melissa Biggs (Biggs.PA @ Xerox.COM)

This document provides a template and instructions for formatting the Lisp Users' module documentation. This template applies primarily to standalone workstation users. Using the Lisp Library module TEdit, and this document, you should be able to create a standard Lisp Users' module for the Lisp Users' manual. This document gives you the written specifications for formatting your document. The specifications are given in the order in which you would most likely use them to format a document, with the basic text and margins described first, then the various levels of headings, then special elements such as page numbers.

RULES FOR CONTENT

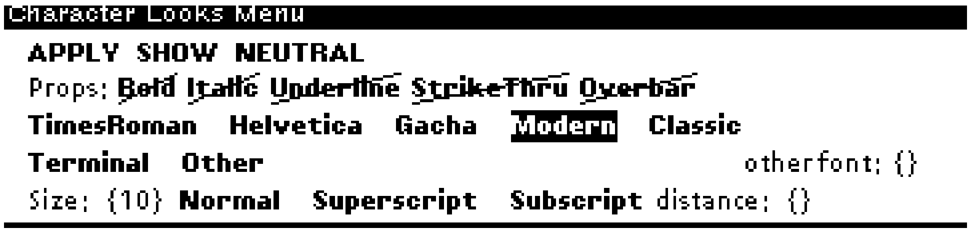
Documentation should always include the name of the module, the name of the author (and Xerox, Arpanet, CSNET or other electronic mailing address, when available—otherwise US mail address), the names of all other Lisp Users' modules required, the names of all files which are part of the module (data files, other Lisp files, etc.), and enough detail to allow someone to effectively use it**.** A sample Lisp Users' template appears at the end of this document.

BASIC SPECIFICATIONS

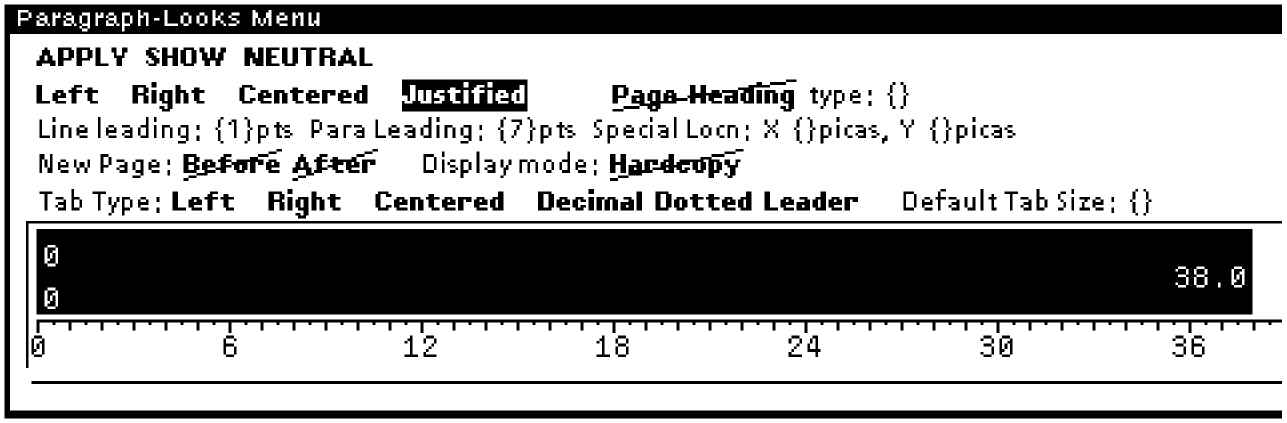
It is wise to apply the specifications for the body text, headings, functions, variables, and page numbers as you write the document.

Font, Type Size, Leading, and Margins

For the text, choose a 10-point Modern font and Apply it to the appropriate text using the Character Looks menu.


Character Looks Menu
APPLY SHOW NEUTRAL
Props: Bold Italic Underline StrikeThru Overbar
TimesRoman Helvetica Gacha Modern Classic
Terminal Other otherfont: {}
Size: {10} Normal Superscript Subscript distance: {}

Then, in the Paragraph Looks menu, set the leading, the spacing between lines of type; the justification; and the left and right margin settings. Set line leading to 1 point and paragraph leading to 7 points. Apply all paragraph looks to the appropriate text.


Paragraph-Looks Menu

APPLY SHOW NEUTRAL

Left Right Centered Justified ~~Page Heading~~ type: {}

Line leading: {1}pts Para Leading: {7}pts Special Locn: X {}picas, Y {}picas

New Page: ~~Before~~ ~~After~~ Display mode: ~~Hardcopy~~

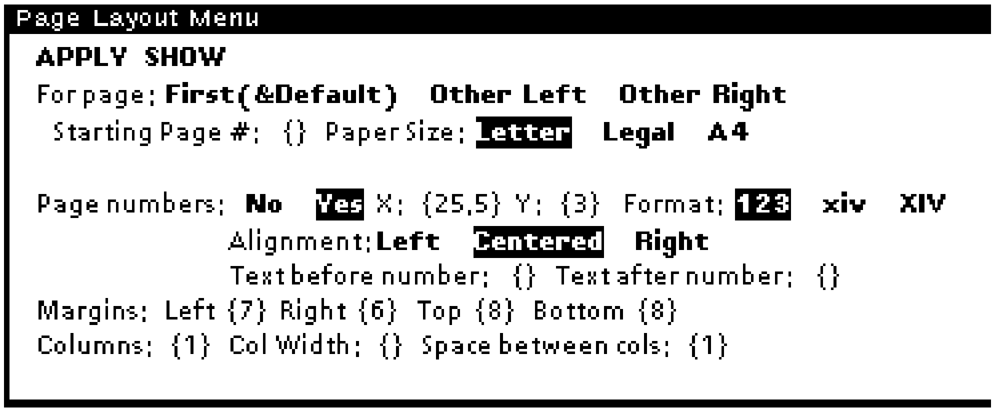
Tab Type: Left Right Centered Decimal Dotted Leader Default Tab Size: {}

0
0
38.0

0 6 12 18 24 30 36

(Because 7 points paragraph leading is all you need, you should use only one carriage return between paragraphs when typing in text.)

Finally, in the Page Layout menu, set the left margin to 7 picas, the right margin to 6, and the top and bottom margins to 8. Apply these to all types of pages (first, other left , and other right).


Page Layout Menu

APPLY SHOW

For page: First(&Default) Other Left Other Right

Starting Page #: {} Paper Size: Letter Legal A4

Page numbers: No Yes X: {25.5} Y: {3} Format: 123 xiv XIV

Alignment: Left Centered Right

Text before number: {} Text after number: {}

Margins: Left {7} Right {6} Top {8} Bottom {8}

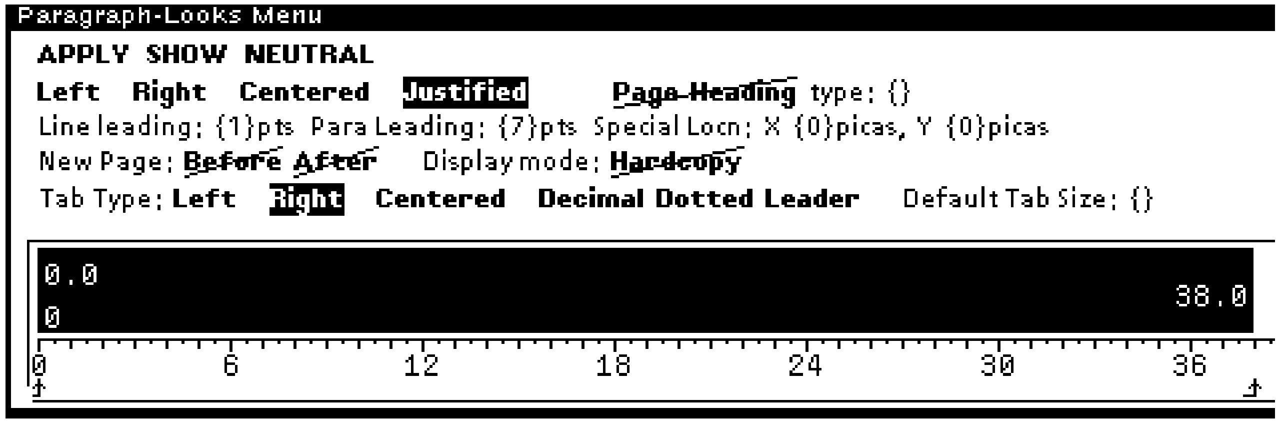
Columns: {1} Col Width: {} Space between cols: {1}

Functions, Variables, and Lisp Code Examples

It is usual to first give the name of a function, then describe its purpose and each of its arguments. When the name of a function is first given, it is set off like this:

(IMAGEFNSCREATE *DISPLAYFN IMAGEBOXFN PUTFN GETFN COPYFN*) [Function]

The Paragraph Looks menu for a function is set up like this:



Paragraph-Looks Menu

APPLY SHOW NEUTRAL

Left Right Centered Justified ~~Page Heading~~ type: {}

Line leading: {1}pts Para Leading: {7}pts Special Locn: X {0}picas, Y {0}picas

New Page: ~~Before~~ ~~After~~ Display mode: ~~Hardcopy~~

Tab Type: Left Right Centered Decimal Dotted Leader Default Tab Size: {}

0.0
0
38.0

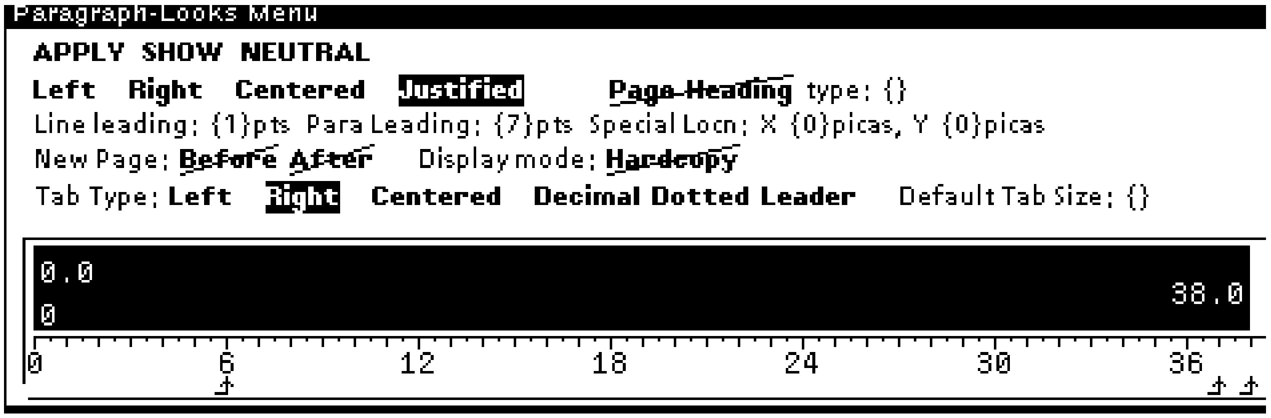
0 6 12 18 24 30 36

The function name is in 10-point regular Modern, all caps. Arguments are in 10-point italic Modern, in all caps, mixed case, or lowercase, as they appear in the system.

If the function description is more than one line long, the runover arguments should be indented under the function name and the word [Function] placed on the last line of the argument list, like this:

(MAKE-ARRAY *INDICESLIST &KEY :ELEMENT-TYPE:* ˆ
*INITIAL-ELEMENT :INITIAL-CONTENTS :ADJUSTABLE:*ˆ
FILL-POINTER :DISPLACED-TO :DISPLACED-INDEX-OFFSET) [Function]

The Paragraph Looks menu should be set as follows to produce this type of argument format:



Paragraph-Looks Menu

APPLY SHOW NEUTRAL

Left Right Centered Justified ~~Page Heading~~ type: {}

Line leading: {1}pts Para Leading: {7}pts Special Locn: X {0}picas, Y {0}picas

New Page: ~~Before~~ ~~After~~ Display mode: ~~Hardcopy~~

Tab Type: Left Right Centered Decimal Dotted Leader Default Tab Size: {}

0.0
0
38.0

0 6 12 18 24 30 36

Long function descriptions should have two points leading between lines. To space them correctly, hold down the meta key when typing the carriage returns so that TEdit breaks the lines without creating separate paragraphs.

Parentheses are in 10-point regular Modern type. The type of definition is in 10-point Modern, caps and lowercase, enclosed in square brackets, and flushed right with a right tab set to 38.0. Note that tabs are used to indent the second and third lines of arguments.

Variables look like functions, except that the word "Variable," enclosed in square brackets, follows the variable name.

Examples of code should be in 10-point Terminal font (but not function names, commands, file names, and the like). If 10-point Terminal is not available on your printer, use 8-point Terminal. Code examples should have two points line leading and no paragraph leading.

Quotation Marks, Bullets, and Dashes

We recommend using TEdit's "expanded abbreviations" to produce professional-looking quotation marks, bullets, em-dashes—used to separate text phrases—and en-dashes (used to indicate inclusive numbers, as in "pages 3–6").

· To produce a bullet, type a lowercase b, select it, and type Control-X.

· To produce an em-dash, type a lowercase m, select it, and type Control-X.

· To produce an en-dash, type a lowercase n, select it, and type Control-X.

HEADS

There are four levels of heads in the Lisp Users' documentation: chapter (level 1) heads, level 2, level 3, and level 4 heads.

Note: A head that falls at the bottom of a page (a "widow") is undesirable. You eliminate a widow by selecting it, then applying the Before option of the New Page command in the Paragraph Looks menu.

The Chapter Head

The chapter head appears at the beginning of the document and identifies it. The heading "Lisp Users' Template," above, is a correctly formatted Lisp Users' chapter head.

When submitting a Lisp Users' module on floppy disk format the chapter head as follows:

Module Name: your Lisp Users' Module (all caps, 12-point bold Modern)

Your name and ARPANET address (if you have one) in 10-point Modern

The Level 2 Head

Level 2 heads identify major sections of a document. The level 2 heads for the Lisp Users' documentation are in 10-point bold Modern, all caps.

The Level 3 Head

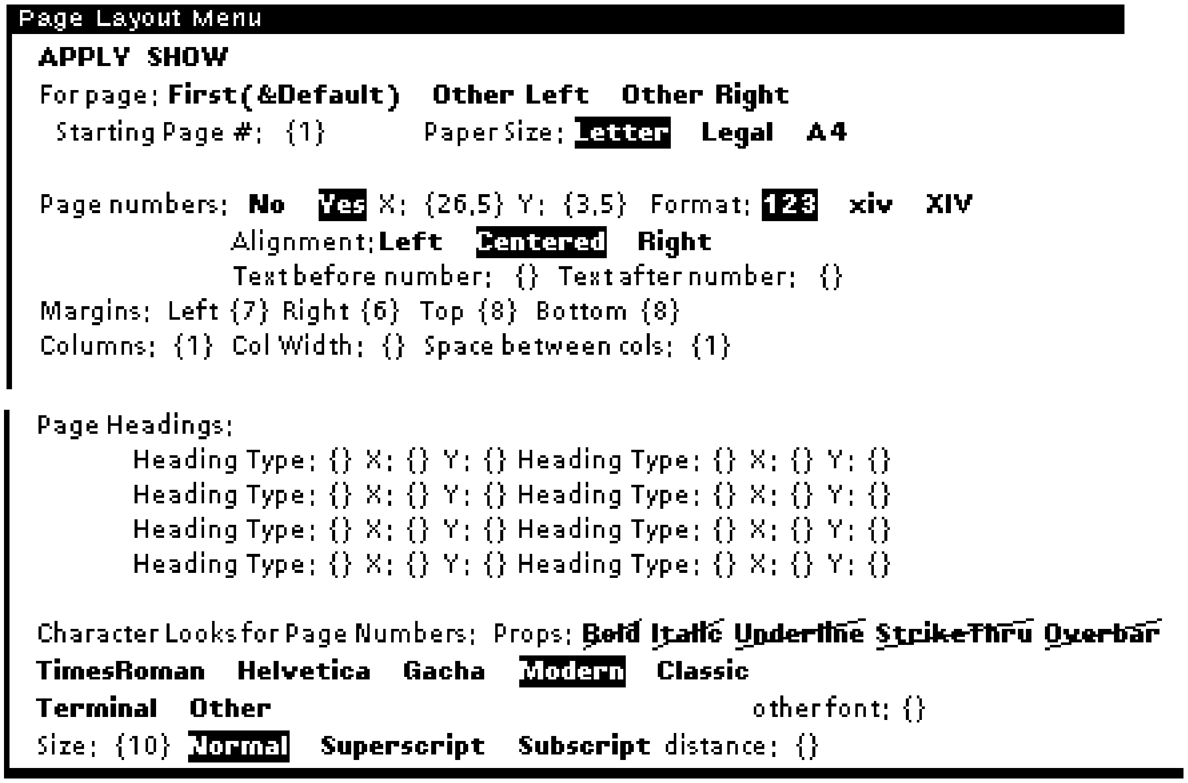
Level 3 heads identify subsections of a document. For the Lisp Users' manual, they are in 10-point bold Modern, caps and lowercase.

The Level 4 Head

Level 4 heads identify the lowest level of subsection in the Lisp Users' documentation. They are in 10-point regular Modern, caps and lowercase, underlined.

PAGE NUMBERS

Page numbers are specified and applied in the Page Layout menu. First, specify the alignment of the page numbers to be centered, with the *X* position being 26.5 picas and the *Y* position 3.5. Then specify the character looks to be 10-point regular Modern. Finally, apply the page numbers to the First(&Default) pages.



Page Layout Menu

APPLY SHOW

For page: **First(&Default)** **Other Left** **Other Right**

Starting Page #: {1} Paper Size: [✓ **Letter**] **Legal** **A4**

Page numbers: **No** [✓ **Yes**] X: {26.5} Y: {3.5} Format: [✓ **123**] **xiv** **XIV**

Alignment: **Left** [✓ **Centered**] **Right**

Text before number: {} Text after number: {}

Margins: Left {7} Right {6} Top {8} Bottom {8}

Columns: {1} Col Width: {} Space between cols: {1}

Page Headings:

Heading Type: {} X: {} Y: {} Heading Type: {} X: {} Y: {}

Heading Type: {} X: {} Y: {} Heading Type: {} X: {} Y: {}

Heading Type: {} X: {} Y: {} Heading Type: {} X: {} Y: {}

Heading Type: {} X: {} Y: {} Heading Type: {} X: {} Y: {}

Character Looks for Page Numbers: Props: ~~**Bold**~~ ~~**Italic**~~ ~~**Underline**~~ ~~**StrikeThru**~~ ~~**Overbar**~~

TimesRoman **Helvetica** **Gacha** [✓ **Modern**] **Classic**

Terminal **Other** otherfont: {}

Size: {10} [✓ **Normal**] **Superscript** **Subscript** distance: {}

After you submit your document, XEROX AIS will add running heads, put in additional formatting, and provide final page numbering to assemble it into the Lisp Users' manual.

MANUAL TEMPLATE

The following page has a sample template for the information required in a Lisp Users' module. Use this template to produce your module documentation.

Module Name:

>>MODULE NAME<<

By: >>Your Name<< (>>Your net address<<)

Uses: >>Other modules necessary to run this one<<

>>Type INTERNAL here if the file is for Internal Use Only<<

This document last edited on >>DATE<<.

INTRODUCTION

>>This paragraph should be replaced by an overview of your module. The information on the previous pages explains the documentation conventions to be used for each Lisp Users' module.<<

MODULE EXPLANATIONS

>>Functions, Variables, and Lisp Code Examples<<

It is usual to first give the name of a function, then describe its purpose and each of its arguments.

Module explanations may have several level headings.

Be sure to include the following information in any module explanations:

• any file dependencies

• definitions of all arguments

• module, function variable, etc. limitations

• a liberal number of examples for all functions, variables, etc.

LISP USERS' TEMPLATE

Updated by: Melissa Biggs (Biggs.PA @ Xerox.COM)

This document provides a template and instructions for formatting the Lisp Users' module documentation. This template applies primarily to standalone workstation users. Using the Lisp Library module TEdit, and this document, you should be able to create a standard Lisp Users' module for the Lisp Users' manual. This document gives you the written specifications for formatting your document. The specifications are given in the order in which you would most likely use them to format a document, with the basic text and margins described first, then the various levels of headings, then special elements such as page numbers.

RULES FOR CONTENT

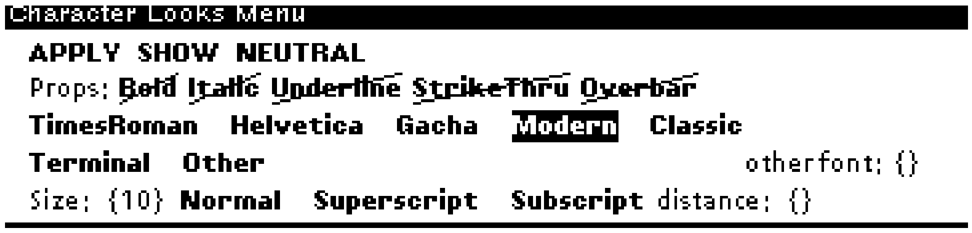
Documentation should always include the name of the module, the name of the author (and Xerox, Arpanet, CSNET or other electronic mailing address, when available—otherwise US mail address), the names of all other Lisp Users' modules required, the names of all files which are part of the module (data files, other Lisp files, etc.), and enough detail to allow someone to effectively use it**.** A sample Lisp Users' template appears at the end of this document.

BASIC SPECIFICATIONS

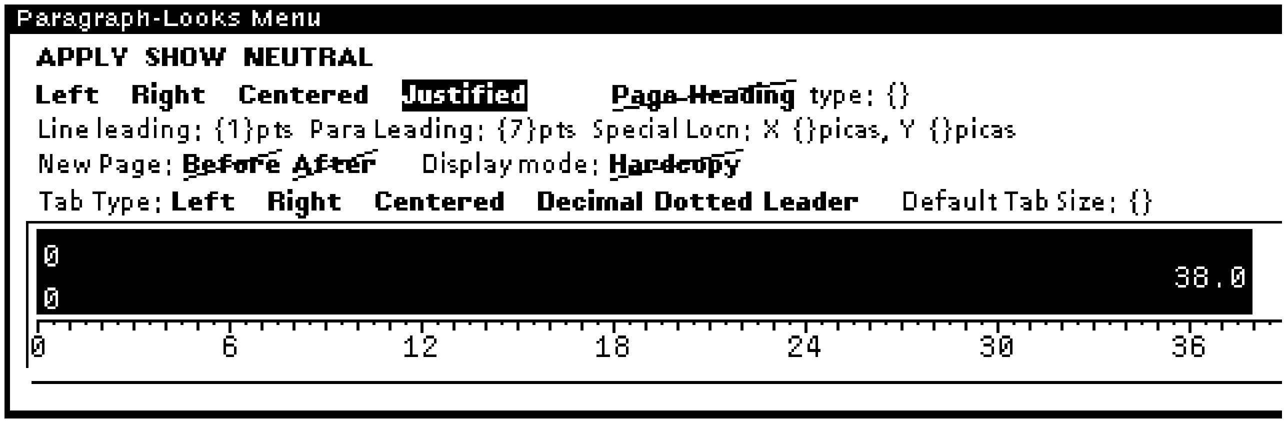
It is wise to apply the specifications for the body text, headings, functions, variables, and page numbers as you write the document.

Font, Type Size, Leading, and Margins

For the text, choose a 10-point Modern font and Apply it to the appropriate text using the Character Looks menu.

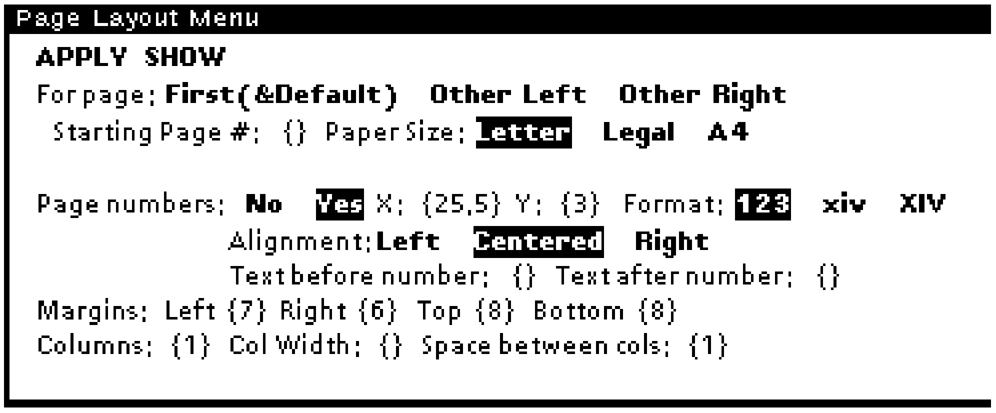

Character Looks Menu
APPLY SHOW NEUTRAL
Props: Bold Italic Underline StrikeThru Overbar
TimesRoman Helvetica Gacha Modern Classic
Terminal Other otherfont: {}
Size: {10} Normal Superscript Subscript distance: {}

Then, in the Paragraph Looks menu, set the leading, the spacing between lines of type; the justification; and the left and right margin settings. Set line leading to 1 point and paragraph leading to 7 points. Apply all paragraph looks to the appropriate text.


Paragraph-Looks Menu
APPLY SHOW NEUTRAL
Left Right Centered **Justified** ~~Page Heading~~ type: {}
Line leading: {1}pts Para Leading: {7}pts Special Locn: X {}picas, Y {}picas
New Page: ~~Before~~ ~~After~~ Display mode: ~~Hardcopy~~
Tab Type: Left Right Centered Decimal Dotted Leader Default Tab Size: {}

(Because 7 points paragraph leading is all you need, you should use only one carriage return between paragraphs when typing in text.)

Finally, in the Page Layout menu, set the left margin to 7 picas, the right margin to 6, and the top and bottom margins to 8. Apply these to all types of pages (first, other left , and other right).

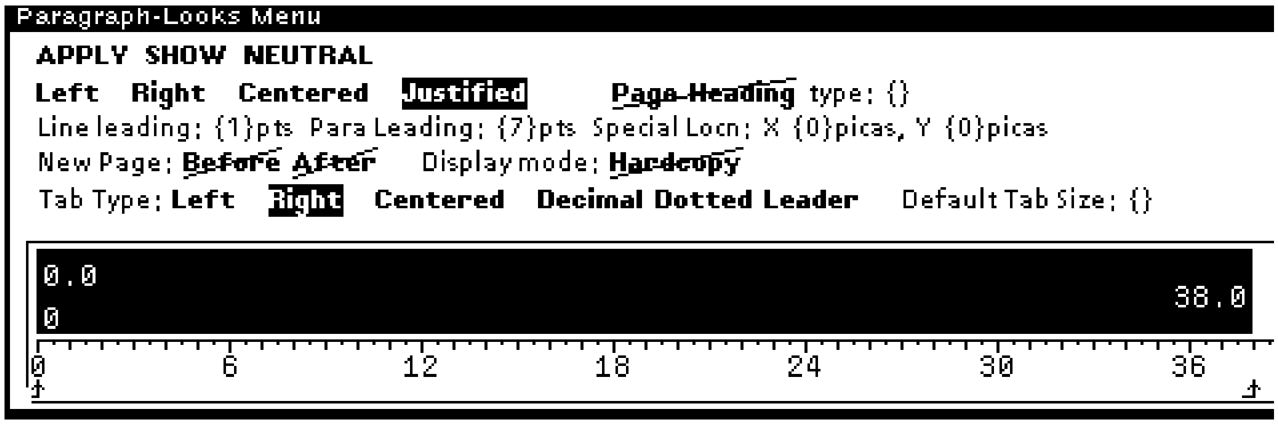

Page Layout Menu
APPLY SHOW
For page: First(&Default) Other Left Other Right
Starting Page #: {} Paper Size: **Letter** Legal A4
Page numbers: No **Yes** X: {25.5} Y: {3} Format: **123** xiv XIV
Alignment: Left **Centered** Right
Text before number: {} Text after number: {}
Margins: Left {7} Right {6} Top {8} Bottom {8}
Columns: {1} Col Width: {} Space between cols: {1}

Functions, Variables, and Lisp Code Examples

It is usual to first give the name of a function, then describe its purpose and each of its arguments. When the name of a function is first given, it is set off like this:

(IMAGEFNSCREATE *DISPLAYFN IMAGEBOXFN PUTFN GETFN COPYFN*) [Function]

The Paragraph Looks menu for a function is set up like this:



Paragraph-Looks Menu

APPLY SHOW NEUTRAL

Left Right Centered [✓ Justified] ~~Page Heading~~ type: {}

Line leading: {1}pts Para Leading: {7}pts Special Locn: X {0}picas, Y {0}picas

New Page: ~~Before~~ ~~After~~ Display mode: ~~Hardcopy~~

Tab Type: Left [✓ Right] Centered Decimal Dotted Leader Default Tab Size: {}

0.0 38.0

0

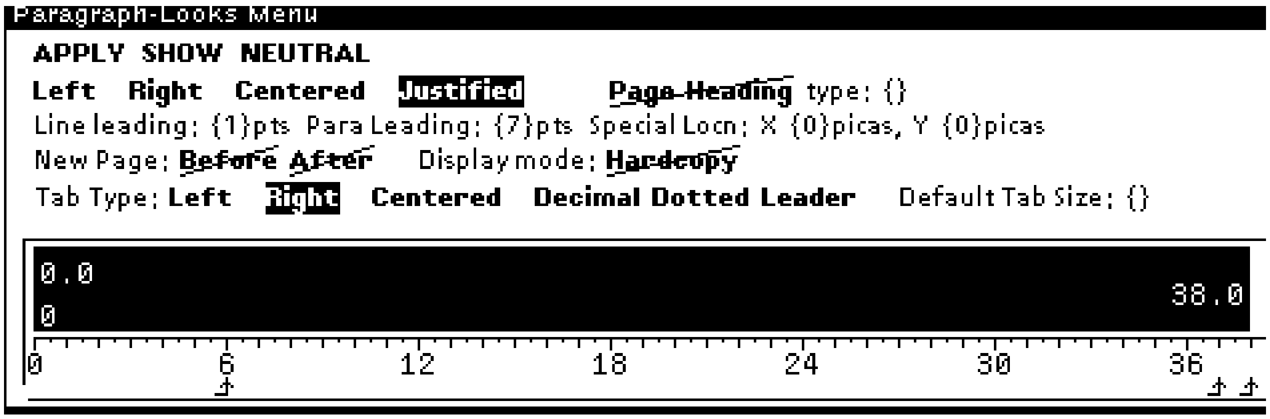
0 6 12 18 24 30 36

The function name is in 10-point regular Modern, all caps. Arguments are in 10-point italic Modern, in all caps, mixed case, or lowercase, as they appear in the system.

If the function description is more than one line long, the runover arguments should be indented under the function name and the word [Function] placed on the last line of the argument list, like this:

(MAKE-ARRAY *INDICESLIST &KEY :ELEMENT-TYPE: ˆ*
INITIAL-ELEMENT :INITIAL-CONTENTS :ADJUSTABLE:ˆ
FILL-POINTER :DISPLACED-TO :DISPLACED-INDEX-OFFSET) [Function]

The Paragraph Looks menu should be set as follows to produce this type of argument format:



Paragraph-Looks Menu

APPLY SHOW NEUTRAL

Left Right Centered [✓ Justified] ~~Page Heading~~ type: {}

Line leading: {1}pts Para Leading: {7}pts Special Locn: X {0}picas, Y {0}picas

New Page: ~~Before~~ ~~After~~ Display mode: ~~Hardcopy~~

Tab Type: Left [✓ Right] Centered Decimal Dotted Leader Default Tab Size: {}

0.0 38.0

0

0 6 12 18 24 30 36

Long function descriptions should have two points leading between lines. To space them correctly, hold down the meta key when typing the carriage returns so that TEdit breaks the lines without creating separate paragraphs.

Parentheses are in 10-point regular Modern type. The type of definition is in 10-point Modern, caps and lowercase, enclosed in square brackets, and flushed right with a right tab set to 38.0. Note that tabs are used to indent the second and third lines of arguments.

Variables look like functions, except that the word "Variable," enclosed in square brackets, follows the variable name.

Examples of code should be in 10-point Terminal font (but not function names, commands, file names, and the like). If 10-point Terminal is not available on your printer, use 8-point Terminal. Code examples should have two points line leading and no paragraph leading.

Quotation Marks, Bullets, and Dashes

We recommend using TEdit's "expanded abbreviations" to produce professional-looking quotation marks, bullets, em-dashes—used to separate text phrases—and en-dashes (used to indicate inclusive numbers, as in "pages 3–6").

· To produce a bullet, type a lowercase b, select it, and type Control-X.

· To produce an em-dash, type a lowercase m, select it, and type Control-X.

· To produce an en-dash, type a lowercase n, select it, and type Control-X.

HEADS

There are four levels of heads in the Lisp Users' documentation: chapter (level 1) heads, level 2, level 3, and level 4 heads.

Note: A head that falls at the bottom of a page (a "widow") is undesirable. You eliminate a widow by selecting it, then applying the Before option of the New Page command in the Paragraph Looks menu.

The Chapter Head

The chapter head appears at the beginning of the document and identifies it. The heading "Lisp Users' Template," above, is a correctly formatted Lisp Users' chapter head.

When submitting a Lisp Users' module on floppy disk format the chapter head as follows:

Module Name: your Lisp Users' Module (all caps, 12-point bold Modern)

Your name and ARPANET address (if you have one) in 10-point Modern

The Level 2 Head

Level 2 heads identify major sections of a document. The level 2 heads for the Lisp Users' documentation are in 10-point bold Modern, all caps.

The Level 3 Head

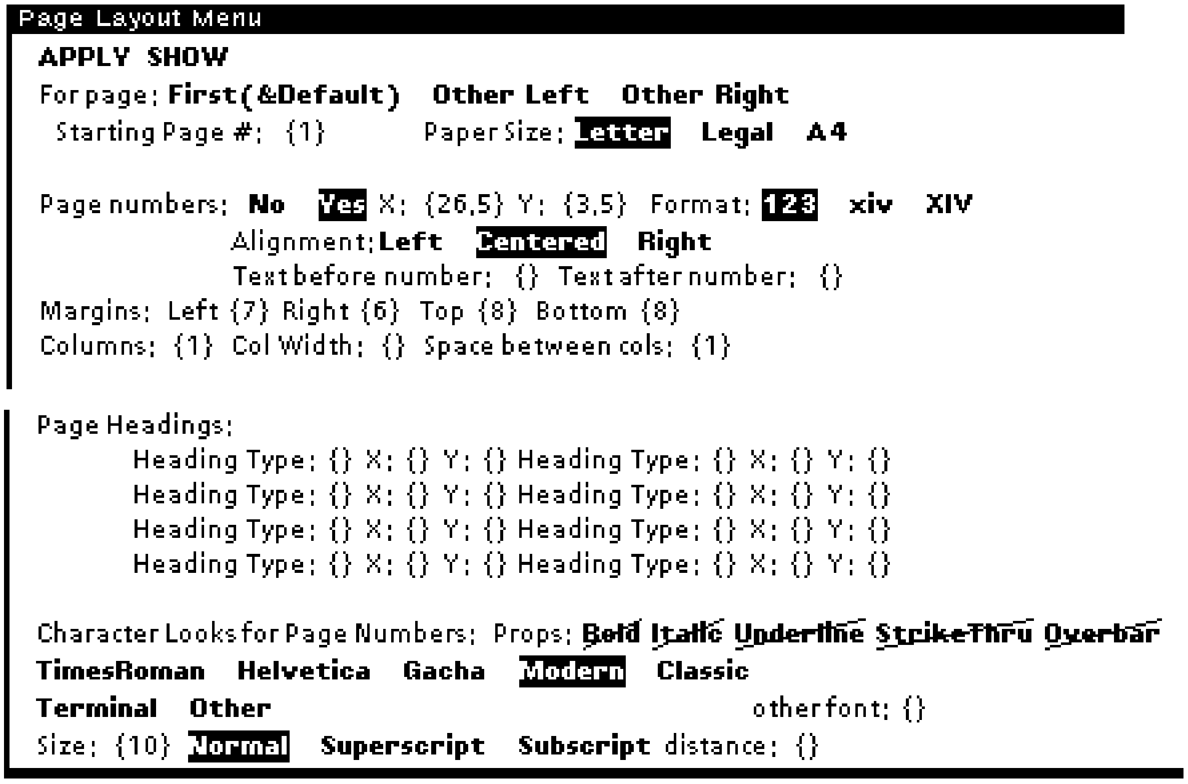
Level 3 heads identify subsections of a document. For the Lisp Users' manual, they are in 10-point bold Modern, caps and lowercase.

The Level 4 Head

Level 4 heads identify the lowest level of subsection in the Lisp Users' documentation. They are in 10-point regular Modern, caps and lowercase, underlined.

PAGE NUMBERS

Page numbers are specified and applied in the Page Layout menu. First, specify the alignment of the page numbers to be centered, with the *X* position being 26.5 picas and the *Y* position 3.5. Then specify the character looks to be 10-point regular Modern. Finally, apply the page numbers to the First(&Default) pages.



Page Layout Menu

APPLY SHOW

For page: **First(&Default)** **Other Left** **Other Right**

Starting Page #: {1} Paper Size: [✓ **Letter**] **Legal** **A4**

Page numbers: **No** [✓ **Yes**] X: {26.5} Y: {3.5} Format: [✓ **123**] **xiv** **XIV**

Alignment: **Left** [✓ **Centered**] **Right**

Text before number: {} Text after number: {}

Margins: Left {7} Right {6} Top {8} Bottom {8}

Columns: {1} Col Width: {} Space between cols: {1}

Page Headings:

Heading Type: {} X: {} Y: {} Heading Type: {} X: {} Y: {}

Heading Type: {} X: {} Y: {} Heading Type: {} X: {} Y: {}

Heading Type: {} X: {} Y: {} Heading Type: {} X: {} Y: {}

Heading Type: {} X: {} Y: {} Heading Type: {} X: {} Y: {}

Character Looks for Page Numbers: Props: ~~**Bold**~~ ~~**Italic**~~ ~~**Underline**~~ ~~**StrikeThru**~~ ~~**Overbar**~~

TimesRoman **Helvetica** **Gacha** [✓ **Modern**] **Classic**

Terminal **Other** otherfont: {}

Size: {10} [✓ **Normal**] **Superscript** **Subscript** distance: {}

After you submit your document, XEROX AIS will add running heads, put in additional formatting, and provide final page numbering to assemble it into the Lisp Users' manual.

MANUAL TEMPLATE

The following page has a sample template for the information required in a Lisp Users' module. Use this template to produce your module documentation.

Module Name:

>>Module Name<<

By: >>Your Name<< (>>Your net address<<)

Uses: >>Other modules necessary to run this one<<

>>Type INTERNAL here if the file is for Internal Use Only<<

This document last edited on >>DATE<<.

INTRODUCTION

>>This paragraph should be replaced by an overview of your module. The information on the previous pages explains the documentation conventions to be used for each Lisp Users' module.<<

MODULE EXPLANATIONS

>>Functions, Variables, and Lisp Code Examples<<

It is usual to first give the name of a function, then describe its purpose and each of its arguments.

Module explanations may have several level headings.

Be sure to include the following information in any module explanations:

• any file dependencies

• definitions of all arguments

• module, function variable, etc. limitations

• a liberal number of examples for all functions, variables, etc.

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.

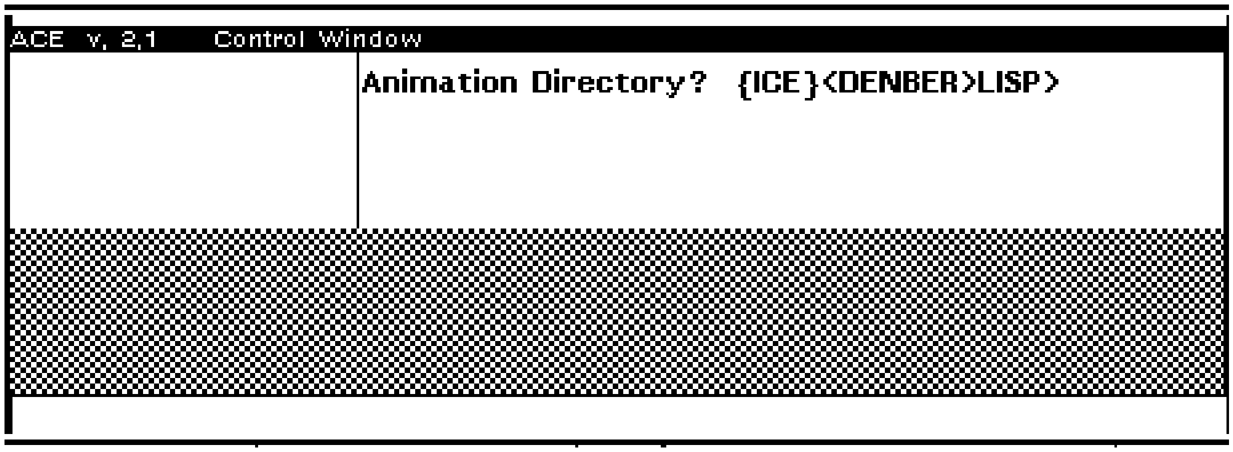
The *ACE Control Window* holds a menu of commands and displays animation state, prompt, and help information. The upper left region in the control window, referred to as the *status region*, tells which frame is currently being displayed (which frame is the current frame); which device (mouse or tablet) 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.

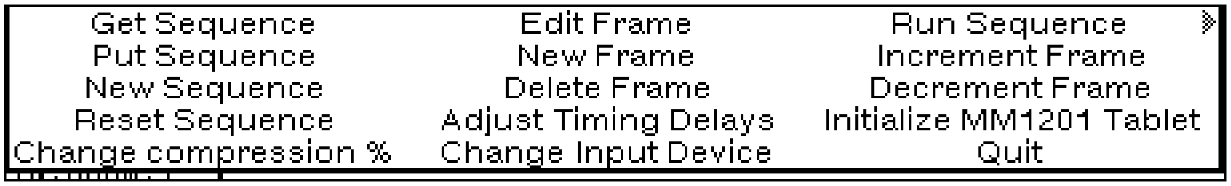

ACE v. 2.1 Control Window
Animation Directory? {ICE}<DENBER>LISP>

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.



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 mimimize 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

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

ACE Maintainer's Notes
Created: 4/29/85
Revisions:

INTRODUCTION

This document is intended for programmers who intend to fix, update or otherwise significantly modify the ACE code. A general description and several specific points about the Animation Compiler and Environment will be brought out This is neither a user's guide nor a guide for animation. A user's guide may be found filed under {ICE}<TURNER>ACE>ACE-USERS-GUIDE; a technical document is filed under {ICE}<WHOKNOWS>ACE>WHOKNOWS.

OVERVIEW

The Animation Compiler and Environment is a program for creating, editing, displaying and storing/retrieving frame-oriented animations. It is based on traditional animation principals; successive frames (each frame usually slightly different from the previous frame) are displayed at a rate fast enough to create the illusion of motion (atleast 10 frames/second; usually more). To the user, the ACE system presents a sequence of whole frames (represented as bitmaps with the same width and height); the analogy being a stack of paper. In reality, however, the program maintains virtual frames. This decreases the storage requirements and allows the very rapid display of frames.

VIRTUAL REPRESENTATION
The virtual frames maintained by ACE are differential frames; that is, a frame only contains the information that is different from the previous frame. By this mechanism, the first frame is a complete frame (complete bitmap) providing the intial information. The second frame, then, is just the differences that exist between the first frame and a hypothetical complete second frame (i.e. just the differences between two bitmaps). And so on with all successive frames. The details of data structures and the compilation process are given below.

TRILLIUM vs STAND-ALONE
Currently, ACE is designed to operate as either a stand-alone program or in the Trillium environment. Conceptually, as a stand-alone, ACE should be considered more procedure oriented. When ACE is called (top level fn: (ACE)), a control window (ACE.CONTROL.WINDOW) is brought up which contains a menu. This window and menu remain up and stay active; once ACE is activated, it becomes a part of the user's current environment. In Trillium, ACE operates more functionally; that is, ACE is called as a one-time function which returns a value (although, ofcourse, it produces side-effects). The control window and menu are only active during the function call and are taken away when the ACE session is concluded. Also, a special animation-run-time function is provided for Trillium to use outside of ACE to run animation item types. At present, Trillium expects all editing (and most loading and storing) of animation sequences to be performed inside ACE.

ORGANIZATION
ACE is organized in a top-down fashion which allows a general to specific description of it. What the components do and how (where appropriate) will now be outlined.

FILES: There are four files to the ACE system (and one utility: RS232): ACE, ACE-MAIN, ACE-EDIT and ACE-PRIM. The divisions exist partly for organizational purposes and partly for ease in locating and working with segments. The ACE file contains some macros and variables, but exists primarily to load in all the necessary files. ACE-MAIN contains all the functions necessary to define, manipulate, run and load/store animations. ACE-EDIT contains all the editing functions (including interfacing to the MM1201 graphics tablet). ACE-PRIM contains the compiler funtions; it is only concerned with compiling two frames to create a virtual frame.

ACE-MAIN
This is the guts of the animation environment. There are four main divisions in MAIN (you can see them by looking at the file coms): Top level fns, Trillium-geared fns, I/O fns, and "helper" fns. In addition, there are several macros and a GLOBALVARS declaration. All the main functions that work with animation sequences are in the first division (including the startup fn: ACE). The I/O section is just for reading and writing files; user input and output fns are in the helper fns section. Control window operation and clipping region fns are also located in the helper fns section.

ACE-EDIT
This segment also consists of four parts (again, note the file coms). The first section provides

entry/interfacing to MAIN and calls to the actual editing routines. LINEART contains all the major fns for line art drawing. The third part contains all the other editing fns (e.g. painting, text, moving, etc.). The last part consists of tablet access fns and helper fns (all MM1201 code is located in this section and the code to read the current input device). The RS232 package is used by ACE-EDIT to read the MM1201 Summagraphics tablet. At present, the RS232 package (RS232.DCOM) is loaded by the ACE file. However, the maintainer should keep aware of changes in this package and its whereabouts.

ACE-PRIM; How the Compiler Operates:
The compilation process is concerned only with compiling two frames at one time (MAIN and EDIT take care of administration). The entry function is ACE.COMPILE.FRAME with arguments: BM.ORIG (the first or original bitmap), BM.CHANGED (the successor bitmap), VERTICAL.BLOCK (defines the height in scanlines for primitive regions of change; the horizontal component is 16 so as to take advantage of the 11xx series word size), and THRESHOLD (the percent as an integer representing the minimum amout of "changed area" allowable in a combination; more on this later). Take note that there are two variables defined in PRIM: ACE.PIXPERWORD (16, just the 11xx word size) and ACE.BITMAP.MASK (a mask used for ignoring extra bits in the last raster word of a bitmap).
The compiler works by comparing each word in BM.ORIG to BM.CHANGED (NOTE: these bitmaps must have the same dimensions) VERTICAL.BLOCK words at a time. If a changed word is found, a region specification is entered on list denoting this "region of change". After the whole area of the bitmaps is checked, the compiler attempts to merge these smallish changed-regions (see ACE.MAX.REGIONS). The algorithm first attempts to combine regions which result in no space wastage (this is always desirable). When no more 100% regions can be formed, the algorithm tests all pairings of primitive regions to find the highest efficiency (i.e. least space waste). If this effeciency is greater than or equal to TRESHOLD, the two regions are combined, otherwise, the algorithm terminates. The region merging process is the slowest aspect of ACE; the problem of merging regions is thought to be NP-complete.

DATA STRUCTURES
- A sequence is of the form: (FRAME FRAME FRAME ...).
- A FRAME is of the form: (DELAY BLITS).
- A DELAY is an integer representing a time delay in milliseconds.
- BLITS is a list: (BLIT BLIT BLIT ...).
- A BLIT is of the form: (BITMAP XCOOR . YCOOR).
A BLIT is essentially a small changed area with information on where it should be placed (relative to the animation). The record defintions for the above structures are in the file ACE. The compiler uses the structures: REGION : (LEFT BOTTOM WIDTH HEIGHT); a modified REGION : (REGION . AREA) for merging; and lists or both of these types.

GLOBALVARS
The following are important global variables with explanations where needed.

- ACE.CONTROL.WINDOW contains the top-level menu and status information.
- ACE.DIRECTORY is a default directory used to store/retrieve files.
- ACE.SEQ.WINDOW the current window where animations are displayed.
- ACE.SEQ.WIDTH and ACE.SEQ.HEIGHT refer to the current sequence.
- ACE.SEQ.WINDOW.XOFF and "."".YOFF the offset in the ACE.SEQ.WINDOW.
- ACE.CURRENT.SEQUENCE points to data which is the current animation sequence.
- ACE.CURRENT.SEQUENCE.NAME for retaining file name information.
- ACE.FRAME.TAIL tail of frames starting one after the current frame.
- ACE.CURRENT.FRAME a tail of frames starting with the current frame.
- ACE.VERTICAL.BLOCK value to use when compiling (see above on compiler).
- ACE.AREA.THRESHOLD for compiling.
- ACE.RUNNING.UNDER.TRILLIUM T if ACE was called by Trillium.
- Various .CURSORs are just cursors.

There are also a GLOBALVARS list of menus in MAIN and EDIT. The approach on menus was that they should be created only once to save both time and space.

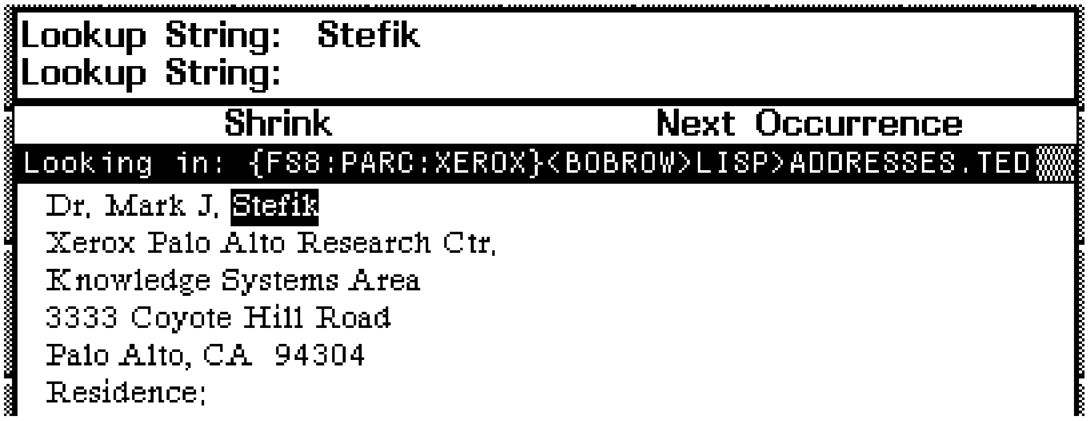
ADDRESSBOOK

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

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.

Type carriage return, ^X, 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 by using the Shrink selection in the title bar .



Lookup String: Stefik
Lookup String:

Shrink Next Occurrence

Looking in: {FS8:PARC:XEROX}<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

```
({PHYLUM}<REGISTRAR>PARCPHONELIST.TXT
 {INDIGO}<REGISTRAR>ISDPHONELIST.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

Starting or Restarting Address Book

Evaluating (MakeAddressBook) will create an address book window and process. This may be useful if you accidentally close the window.

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, click middle button in the title of the ADDRESSBOOK window, and select the option "Edit file named in window title". A TEDIT process editing the file will be set up. This process is independent of the lookup process. To select the file to be edited, rolloff the above item, and select "Select file to edit". A menu of files used by the Lookup process will be presented to you. Selecting one will cause that file to be edited.

To make editing changes visible to the lookup process, PUT the file in TEDIT; when it is done, recache the the file in core. To recache just the file edited, (the one specified in the title bar of the window), select the option "Recache file named in window title" in the middle button title bar menu. You can recache all files by selecting the option "Recache all files" in the title menu (a subselection of the item "Recache file named in window title".

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.

Deleting a file from the List of Files

To delete from the list of files being used for lookup, select the option "Delete file from list" in the title bar menu. This file will be deleted from the list of files to be searched.

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 your 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 costmetic 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, independant 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 interesect 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 *iregion*s 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:

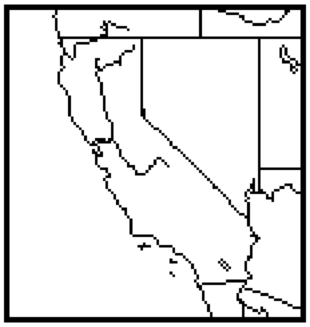
```
(WINDOWPROP (WHICHW) 'IREGIONSLIST IRS)
(WINDOWPROP (WHICHW) 'BUTTONEVENTFN 'IN.CURSOR.REGION)
```

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:

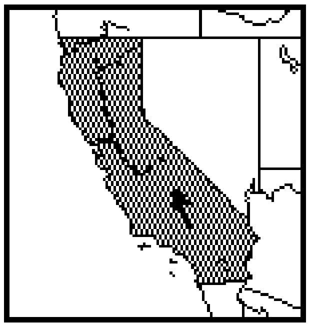
1. Open a window...about 1/4 of a screen.
2. Use the paint function provided when you right-button in the window and paint a picture.



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)))))
```

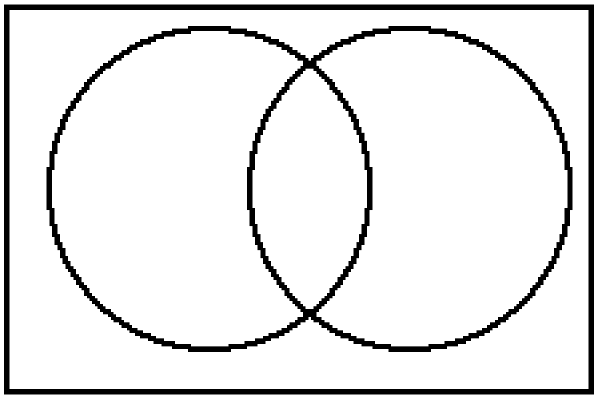
Span the cursor out over another state/region and repeat steps 3-5 above. When you button in this IREGION, the IREGION will temporarily shade black, and call the RINGBELLS function. Note that like menu selection, the function is called only when you release the button within the region. If the mouse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

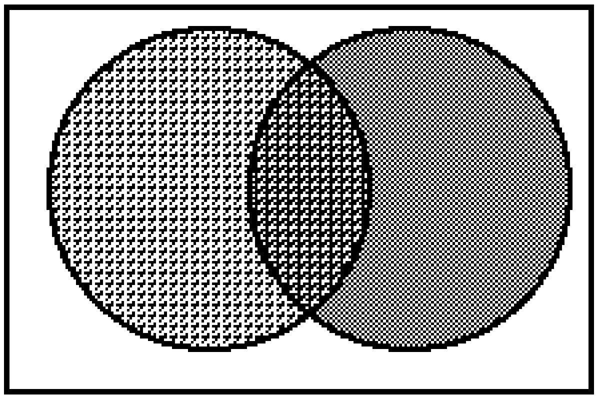
Intersecting Iregions

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.

AISBLT

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

This document last edited on September 21, 1988.

INTRODUCTION

The AISBLT module provides a fast(er) interface for reading AIS format files into Lisp bitmaps. It does not provide all the arcane features found in the READAIS module.

CLIENT INTERFACE

The functions provided by the AISBLT module which are intended to be used by clients are

(AISBLT.BITMAP *FILE SOURCE-LEFT SOURCE-BOTTOM DESTINATION DESTINATION-LEFT DESTINATION-BOTTOM WIDTH HEIGHT HOW FILTER*) [Function]

The *SOURCE-LEFT*, *SOURCE-BOTTOM*, *DESTINATION*, *DESTINATION-LEFT*, *DESTINATION-BOTTOM*, *WIDTH*, and *HEIGHT* arguments are interpreted in the same way as the corresponding arguments to BITBLT. *FILE* is either an open stream, or a filename. If a filename is provided it will be passed to FINDFILE, which searches the directories specified by the special variable

AISDIRECTORIES [Variable]

which should be a list of directories where the AIS file is likely to be found.

The argument *HOW* should be one of the atoms FSA, :FSA, TRUNCATE, :TRUNCATE. *HOW* is only applicable in the cases where the source and destination are a different number of bits per pixel (source bpp > destination bpp). If *HOW* is not specified, it defaults to FSA. FSA indicates that the source should be reduced to the bits per pixel of the destination by applying the Floyd-Steinberg dithering algorithm, as described in Newman & Sproull, Principles of Interactive Computer Graphics, pg. 226. TRUNCATE indicates that only the high order bit(s) of the source should be used.

The function

(AISFILEHEADER *STREAM*) [Function]

Can be used to determine whether a file has a well formed AIS header, and what the attributes indicated in the header are. The result of the function is a property list describing the AIS attributes:

:RASTER [Key]

The :RASTER property will always be present. The value is also a property list

> :SCAN-COUNT [Key]
>
> An integer value indicating the number of scan lines in the image
>
> :SCAN-LENGTH [Key]
>
> An integer value indicating the number of pixels in a scan line of the image

:SCAN-DIRECTION [Key]

An integer, indicating the direction of the scan. Scan direction 3 is top to bottom, left to right, and is the only scan direction that this package will deal with at this time.

:SAMPLES-PER-PIXEL [Key]

An integer, indicating the number of samples per pixel. This package will only deal with files having one sample per pixel at this time.

:CODING-TYPE [Key]

An unsigned integer indicating the coding type of the raster image. A value of 1 indicates uncompressed array format, and is the only type recognized by this package at this time. For convenience, the constant

AIS-RASTER-CODING-UCA [Constant]

is bound to the value 1. If the raster coding types are extended, more constants will be defined.

The rest of the properties are coding type dependent. For the AIS-RASTER-CODING-UCA file, the following properties are present:

:BITS-PER-SAMPLE [Key]

An unsigned integer, indicating number of bits per sample

:WORDS-PER-SCAN-LINE [Key]

An unsigned integer, indicating how many 16 bit words form a single scan line of the image.

:SCAN-LINES-PER-BLOCK [Key]

A signed integer, indicating how many scan lines are present before there is block padding. A value of -1 indicates no blocking.

:PADDING-PER-BLOCK [Key]

A signed integer, indicating how many padding words per block. A value of -1 indicates no blocking.

:PLACEMENT [Key]

The placement property is optional. The value is a property list with keys :LEFT, :BOTTOM, :WIDTH, and :HEIGHT. The values are unsigned integers.

:PHOTOMETRY [Key]

The photomety property is optional. The value is a property list with keys :SIGNAL (integer), :SENSE (integer), :SCALE (integer), :SCALE-A (pair of integers), :SCALE-B (pair of integers), :SCALE-C (pair of integers), :SPOT-TYPE (integer), :SPOT-WIDTH (integer), :SPOT-LENGTH (integer), :SAMPLE-MIN (integer), and :SAMPLE-MAX (integer).

A complete description of the meaning of the photometry parameters can be found on page 38 of the AIS format description, filed on {indigo}<altodocs>aismanual.press.

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.

AUTOSAMEDIR

By: Mitchell L Model
473 Edgell Road
Framingham, MA 01701

Uses: SAMEDIR.LCOM

INTRODUCTION

This package is an extension to the SAMEDIR Library package so that if AUTOSAMEDIRFLG is non-NIL, MAKEFILE will automatically switch to the directory its file argument was originally made to instead of invoking the SAMEDIR dialogue. (It uses the FILEDATES property of the file.)

Notice and Acknowledgement: This package was developed while the author was an employee of Applied Expert Systems, Inc. (Apex), Cambridge, MA. The author thanks the company for its support and assumes full responsibility for the contents and maintenance of this package.

SOFTWARE REQUIRED

AUTOSAMEDIR.LCOM

SAMEDIR.LCOM

FIXES

Extension to SAMEDIR package so that if AUTOSAMEDIRFLG is non-NIL, MAKEFILE will automatically switch to the directory its file argument was originally made to instead of invoking the SAMEDIR dialogue. (It uses the FILEDATES property of the file.)

AUXMENU

By: David Newman (Newman.pasa @ Xerox.COM.ARPA)

AUXMENU is a Lispusers package that creates a middle-button background menu. This menu acts like the right-button background menu that exists in any Interlisp sysout in most respects. The menu includes commonly used Interlisp functions that require no arguments to be useful.

Global Variables

MiddleButtonBackgroundMenuCommands [Variable]

DefaultMiddleButtonBackgroundMenuCommands [Variable]

MiddleButtonBackgroundMenu [Variable]

MiddleButtonBackgroundMenuCommands is a list of MENU items (in the same format described in the Interlisp Reference Manual) which is used to create the middle button background menu. DefaultMiddleButtonBackgroundMenuCommands is the default value of MiddleButtonBack-groundMenuCommands. Individual users may reset MiddleButtonBackgroundMenuCommands, or they may add to or change it. If the user changes this variable, MiddleButtonBackgroundMenu should be set to NIL. All variables are initialized by the package as it is loaded. All necessary interaction is performed in the promptwindow, and the result of any menu item is printed to the promptwindow if it is non-nil.

Default Menu Items

Login - This item performs a (LOGIN) via the promptwindow.

Greet - This item performs a (GREET).

Logout - This item does a (LOGOUT).

Cleanup - This item performs a (CLEANUP) via the promptwindow.

Reclaim - This item executes (RPT 5 (QUOTE (RECLAIM))).

Closeall - This item closes all currently open files by performing (CLOSEALL).

Open Files - This item lists the currently open files to the promptwindow using (OPENP).

Connect - This item prints the currently connected directory to the promptwindow. The subitems of this item make the connected directory the one shown as the item. The ’Default’ subitem connects the system to the value of the variable LOGINHOST/DIR. The ’Other’ subitem prompts the user for a directory name, makes that the connected directory, and adds it to the menu.

VMem Size - This item prints the current size of the virtual memory file (in pages) to the promptwindow. It uses (VMEMSIZE).

Free Pages - The number of free pages in device DSK are printed to the promptwindow. (DISKFREEPAGES) is the function that provides this information.

Disk Partition - This item prints the name of the current partition to the promptwindow. The printed value is the result of calling (DISKPARTITION).

Volume Display - Turns the volume display window on via (VOLUMEDISPLAY ’ON).

Default Printers - Types the value of DEFAULTPRINTINGHOST to the promptwindow.

File Changes - Lists the result of (FILEPKGCHANGES) to the promptwindow.

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

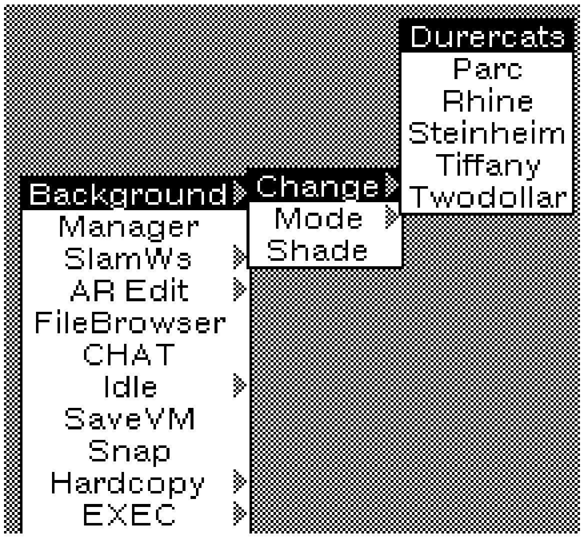
This document last edited on September 8, 1988.

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.REFECT *BITMAP*) [Function]

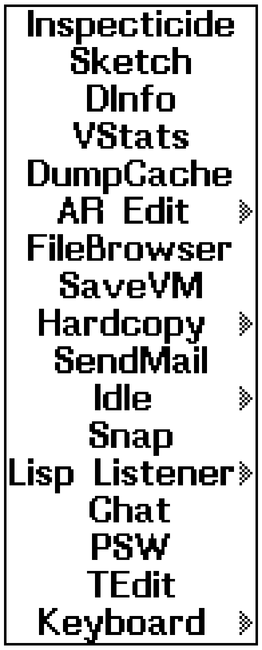
Returns a screen-sized tiled bitmap such that each tile is a reflection of those adjacent to it and such that the center tile is a copy of BITMAP.

BACKGROUNDMENU

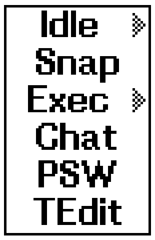
By: Mike Dixon
New Owner: Burwell (Burwell.pa@Xerox.com)

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.

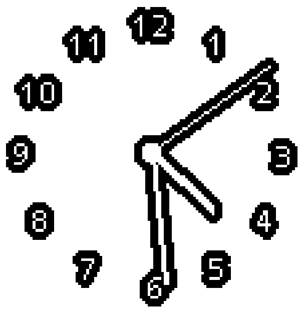
When any of the above functions (except BkgMenu.add.item) require you to specify an item, you can usually just give a string with the menu entry (or an atom, which is coerced to a string). The case has to be correct, and blanks have to be in the right place. The function will do a breadth first search 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".

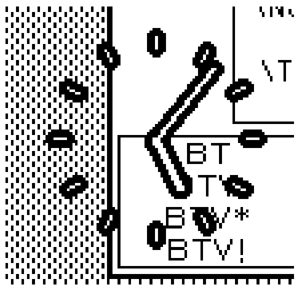
The item argument to BkgMenu.add.item is a standard menu item, i.e. a list of (label form help.string).

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.

BICLOCK

By: Bernt Nilsson (Bernt Nilsson:Ida:LiTH or BKN%LiUIDA.UUCP@Seismo.ARPA)





INTRODUCTION

BICLOCK, a realtime screen clock, with hour, minute and second hands. It behaves as a kind of icon, in that it is partly transparent. The snapshot to the right above shows that the greater part of the window's area is transparent, in this case the gray background and part of the PSW window show through. The clock-image may be either black or white, and the shadow around the image-parts the opposite color: The reason for having a shadow is to make the clock easier to read. Markers and/or digits may be chosen.

Seconds

The seconds hand is optional. When on, the clock process consumes a lot of time, but because (1) it blocks very frequently and (2) shuts itself off temporarily, if the "load" is higher than usual, that does not seem to irritate at all. The "load" is measured by the continuous average of time spent in block, i.e. each round robin. A "usual load" is computed by a second level of average over the "load" value. The "usual load" is limited by upper and lower "reasonable" constants. If "load" is more than 10% above "usual load" then the seconds hand is shut off. This is a rather heuristic method, but works reasonably well. When the seconds hand is shut off, the image is updated approximately once per minute.

WINDOW COMANDS

Most window commands are applicable. The clock will preserve its "square" form when shaped. More details of the hour/minute markers/digits will appear only when the window is large enough, less if the window is smaller. The digits' fonts are chosen among fonts already "in core".

Left button

Left buttoning the window will print the current date (and alarmtime) in the promptwindow. Holding down left shift key while buttoning the window will copyselect current date, by BKSYSBUF.

Middle button, Alarm

There is a simple alarmclock facility built into the clock. That and some other options may be accessed by middlebuttoning the window and selecting the appropriate command from the menu that pops up.

Choosing the command "Set Alarm" will change the clock into showing current alarmtime and attach an adjust menu below the clockwindow. If no alarm time is known, current daytime plus 1 minute is used as default. Any change to the alarmtime is shown both by the clock and printed in the promptwindow. Buttoning "OK!" in the attached menu exits the adjust mode. When the alarmtime is reached, you will be aware of that fact... to shut the alarm off, use the "Alarm Off" in the popup menu.

FUNCTIONALITY

One clockwindow is usually created automatically when the file is loaded. More clockwindows may be created by the following function call:

(BICLOCK *props*) [Function]

Creates a new window. Value is the new window. The call may be done either with a free property list as first arg or spread. See properties below. The different properties are also controlled by the variables below. It is allowed to have more than one clock running, but see "bugs" below.

When the BICLOCK file is loaded, the following variables are initialized, some through INITVARS, so that you can set them before loading the file:

BICLOCKUSERPROPS [Variable]

Change this to customize the overall behavior of the clock. Value must be a free property list. Defaults to NIL.

BICLOCKDEFAULTPROPS [Variable]

These are the default properties, don't modify this, modify BICLOCKUSERPROPS above.

Properties are searched for in the following order.

1. those specified in the function call.
2. those specified by the BICLOCKUSERPROPS variable.
3. those specified by the BICLOCKDEFAULTPROPS variable.

BICLOCKINITIALPROPS [Variable]

Modify this if you like some specific property for the initial clock. BICLOCK is called with these properties when the file is loaded, defaults to NIL. If you want no initial clock, set it to (CREATE NIL).

BICLOCKWINDOW [Variable]

The window of the clock that is created when the file is loaded.

BICLOCKIDLEPROPS [Variable]

(IDLE.BICLOCK) [Function]

BICLOCK is called with BICLOCKIDLEPROPS from the Idle function IDLE.BICLOCK, defaults to start bouncing at center of screen.

Properties

Recognized properties and allowed values are:

SECONDS	T (default), if seconds hand shall be used, or NIL, if no seconds
COLOR	interior color, one of WHITE (default) or BLACK
MARKS	NIL (default), if no marks should be used, or one of HOURS, HOUR&MINUTE, 3/6/9/12 or a modulo number

DIGITS	NIL, if no digits, or one of HOUR (default), 3/6/9/12 or a modulo number
DIGITFORMAT	one of ARABIC or ROMAN
CHIME	NIL (default), no chime,or one of HOUR, QUARTER or a modulo number
ALARM	NIL (default), no alarm, or a the standard string representation of a date
SIZE	a number describing both width and height of the clock window, defaults to 119
HORIZONTAL	generic place on screen, one of LEFT, CENTER, RIGHT (default), or a number to specify left border of window
VERTICAL	generic place on screen, one of BOTTOM, CENTER, TOP (default), or a number to specify bottom border of window
REGION	if nonNIL, a region, overides SIZE, HORIZONTAL and VERTICAL for clockwindow
WINDOW	if nonNIL, a window, overides REGION, SIZE, HORIZONTAL and VERTICAL
CREATE	T (default), create a clock, or NIL, do not create, probably only useful if you want no initial clock...

The nonformal meaning of the Modulo numbers mentioned above are: "1" means: on each possible place. "2" means: on even places and so on...

KNOWN BUGS:

1. If the window is reshaped when in "set alarm" mode, the whole window group will be square.
2. Running more than one clock concurrently with seconds hand on, may behave a bit round robin because the load sensor is local for each clock and "senses" the other clocks.
3. Overlapping two clock windows creates a funny image in the intersecting area.

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).

XEROX

Xerox Corportation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

Bitmap-Gallery

Sampler and Documentation for bitmaps, especially those useful as screen background.

Eventually I'll rationalize the names and packages. Most of these bitmaps have been snapped from the screen and are smaller than they might appear. They usually look better on screen than they do on paper. Please send suggested additions to Foster.PA.

WARNING: This file is only usable in Lyric.

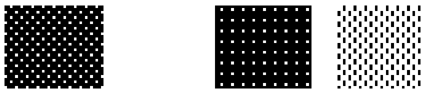
Shades

Author: Gregg Foster et al <Foster.pa>
Filed on: {FS8:PARC:XEROX}<FOSTER>LISP>USERS>BITMAP-GALLERY.TEDIT;5
Last filed: March 23, 1988 18:07 PDT (Wednesday)

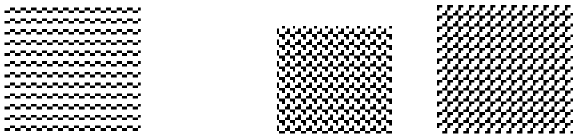
Some Basic Shades

Use (IL:EDITSHADE) to create your own.

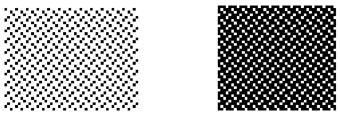
IL:PLAINSHADE, IL:GRAYSHADE, IL:GRAYSHADE1, IL:GRAYSHADE2



IL:GRAYSHADE3, IL:GRAYSHADE4, IL:DEFAULTSCREENSHADE

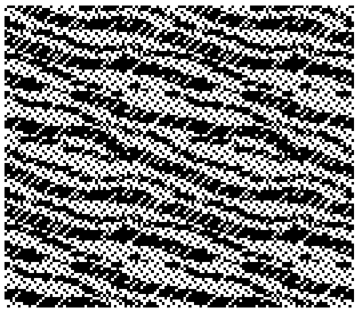
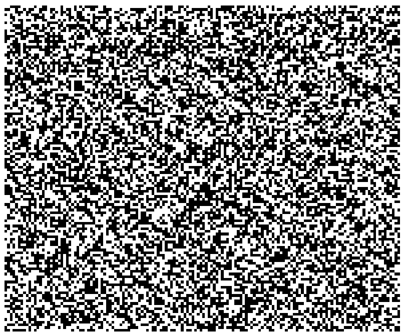


IL:WAVE-TEXTURE, IL:WAVE2-TEXTURE, IL:MESH-TEXTURE

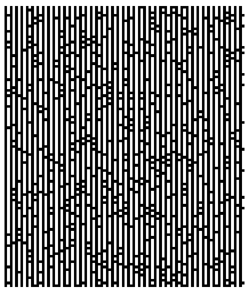
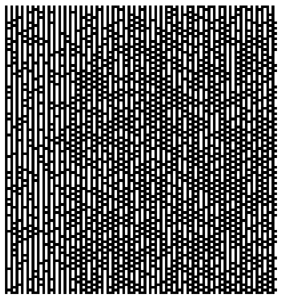


IL:DI-TEXTURE, IL:DARK-DI-TEXTURE

From Gregg Foster

XCL-USER::*TESSEL-BM*, XCL-USER::*RANDOM-BM*

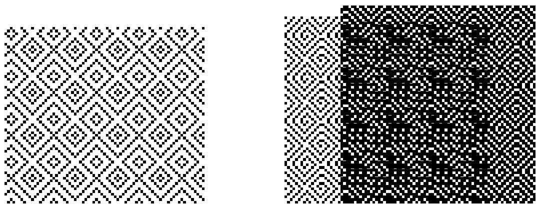



XCL-USER::*GRANITE-LIGHT-BM*, ...-MEDIUM-..., ...-DARK-...

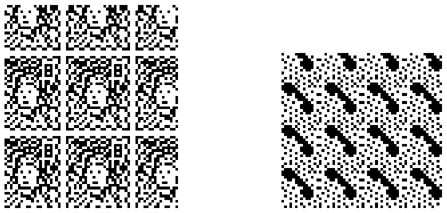
From {PHYLUM}<Foster>Lisp>Users>GRANITE, using the function il:|MakePseudoRandomBitmap| you get bitmaps that look something like the above (they come in three shades, LIGHT, MEDIUM, and DARK (symbols for 3, 2, 1, respectively); and large variety. The three bitmaps above are examples of the kind of thing you get.

[Example usage: (setq my-bitmap (il:|MakePseudoRandomBitmap| NIL 64 64 3), makes a 64x64 light PseudoRandom bitmap.]

From Stanley's Tool Works



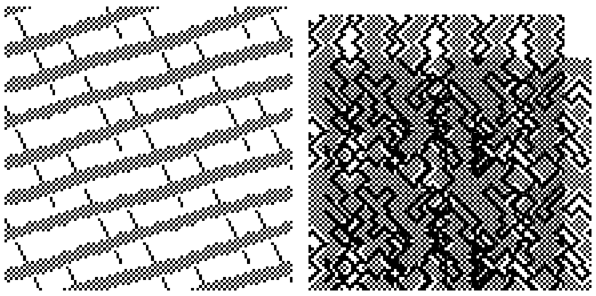
IL:LIGHTWALLPAPER, IL:WALLPAPER, IL:DARKWALLPAPER



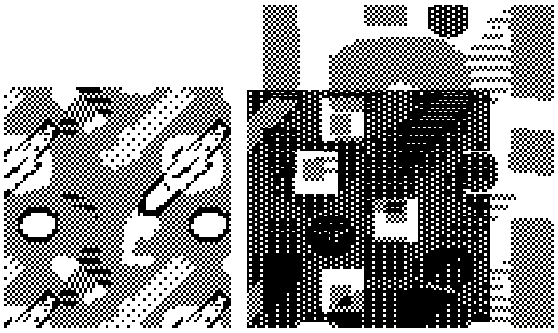
IL:*STAMP-BITMAP*,   IL:*PHONE-BITMAP*

Wickberg

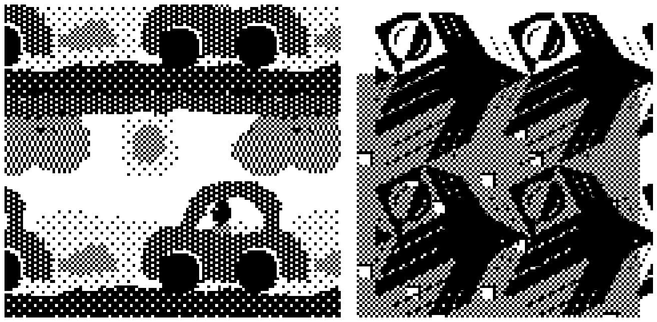
From Andreas Wickberg



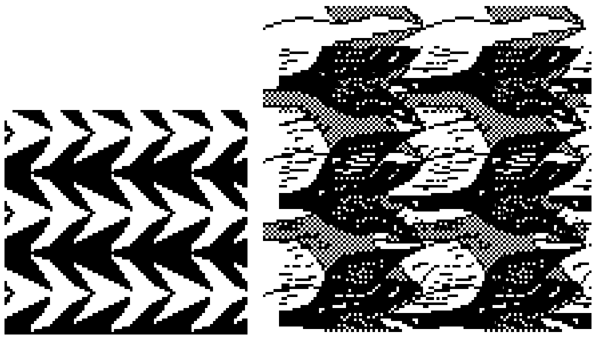
IL:AVANTBACKGROUND0, ...1, ...2



IL:AVANTBACKGROUND3, ...4, ...5

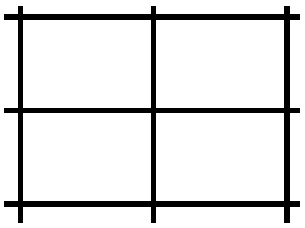
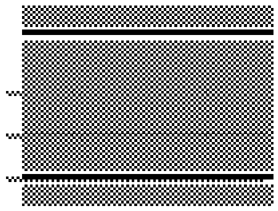


IL:AVANTBACKGROUND6, ...7, ...8

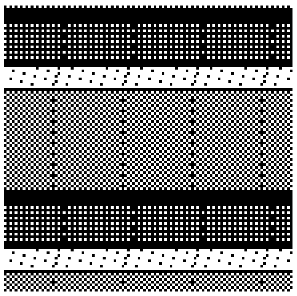
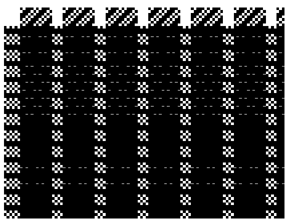


IL:AVANTBACKGROUND9, ...10, ...11

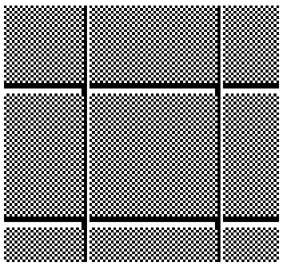
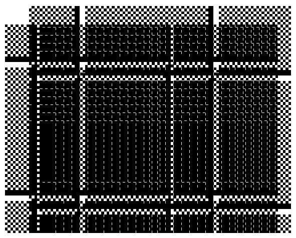
From Stu Card

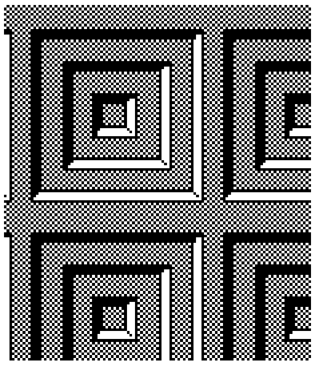
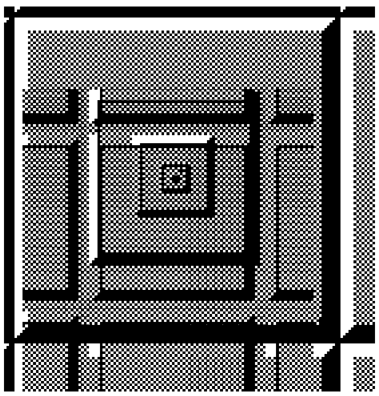
IL:ROOM.BM, IL:LINE1.BM, IL:LINE2.BM,

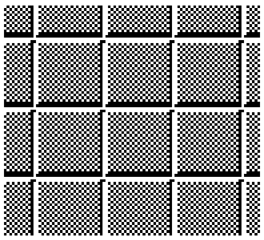
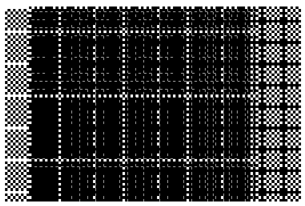
IL:LINE3.BM, IL:LINE4.BM, IL:LINE5.BM

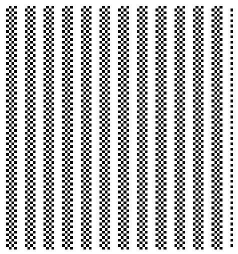
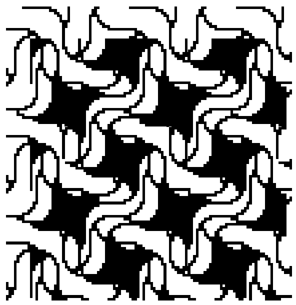
IL:SQUARE1, IL:SQUARE2, IL:SQUARE3.BM, ■

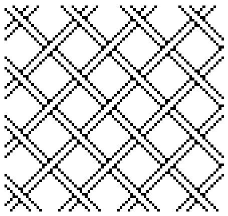
IL:SQUARE4.BM, IL:SQUARE5.BM, IL:SQUARE6.BM

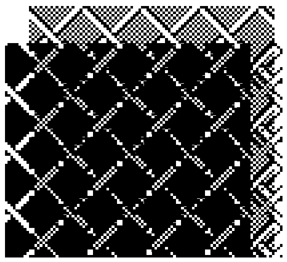



IL:SQUARE7.BM, IL:SQUARE8.BM, IL:SQUARE9.BM

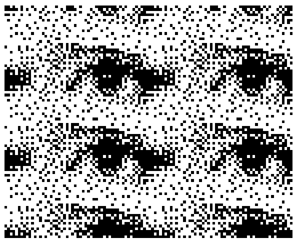
IL:SHIRT1.BM, IL:CURLY, IL:CURLY1





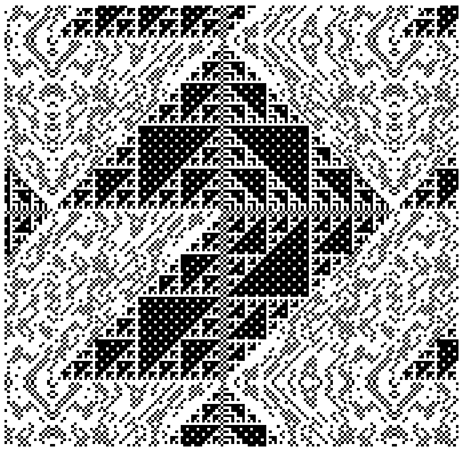
IL:WOVEN, IL:WOVEN1, IL:WOVEN2, IL:WOVEN3

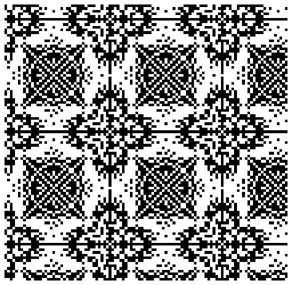
From Harley Davis



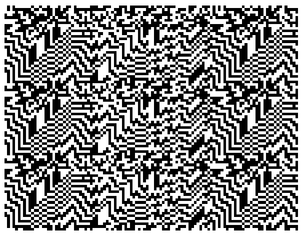
XCL-USER::*EYE-BM*

From John Corbett



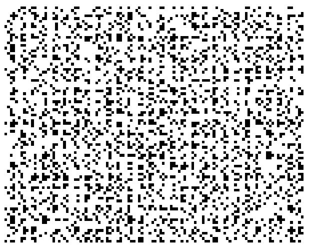
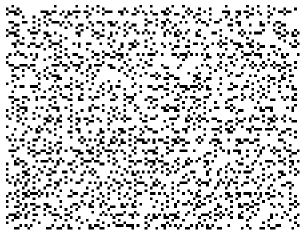


XCL-USER::*FRACT-BM*   XCL-USER::*MANDALA-BM*

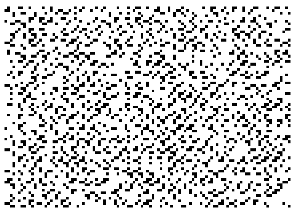
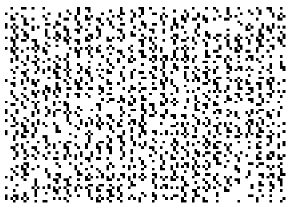


XCL-USER::*STATIC1-BM*   XCL-USER::*STATIC2-BM*

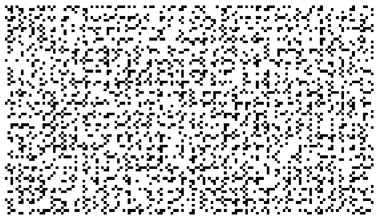
Fabrics (from John Corbett and Gregg Foster)

XCL-USER::*TWEED-BM* XCL-USER::*CHAMBRAY-BM*

XCL-USER::*CANVAS-BM*   XCL-USER::*CORDUROY-BM*

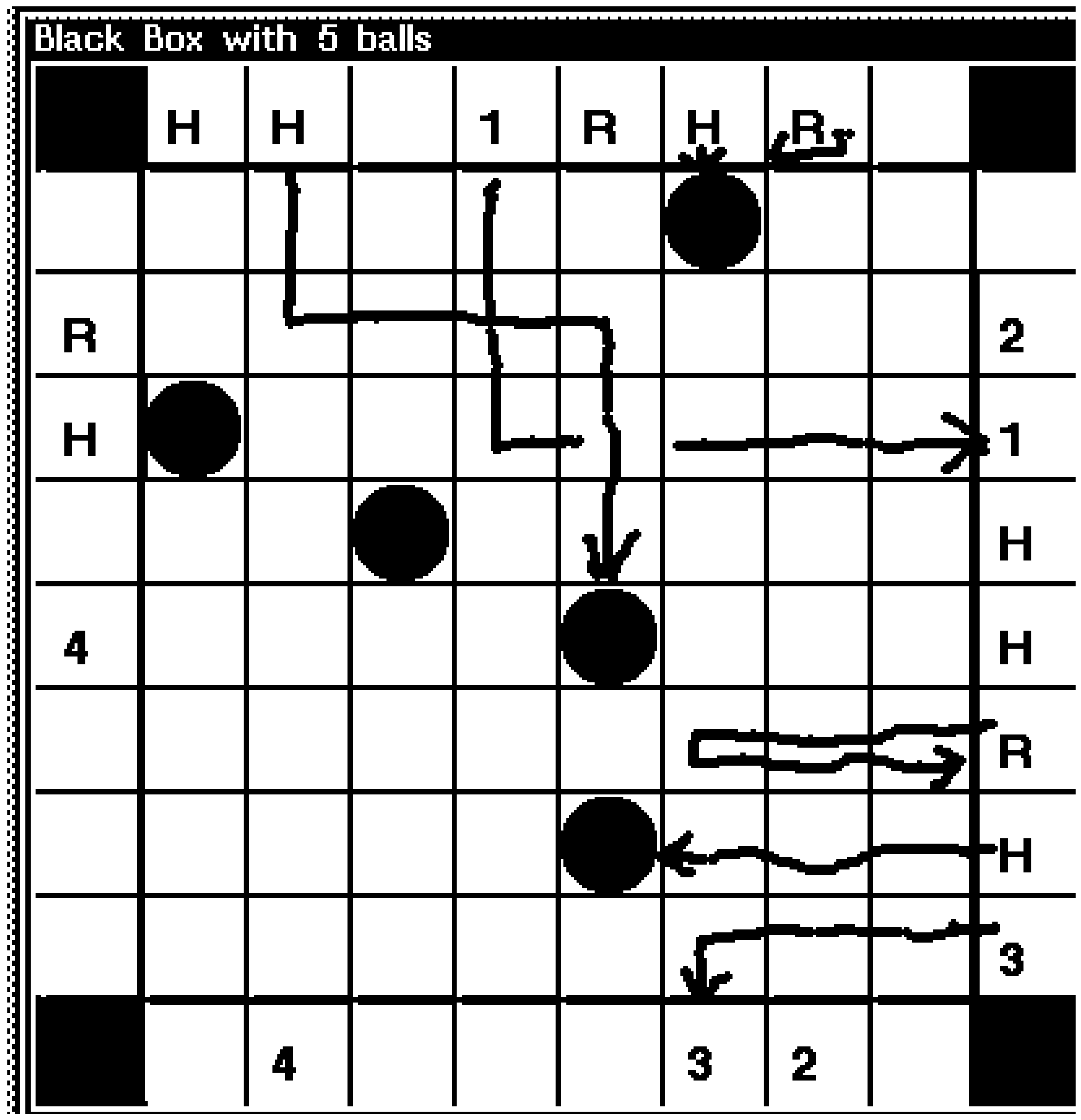


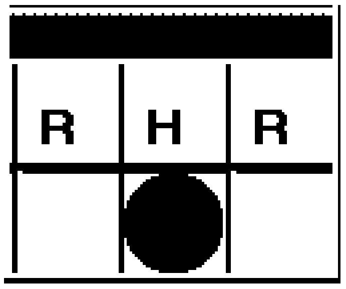
XCL-USER::*SEERSUCKER-BM*   XCL-USER::*BURLAP-BM*

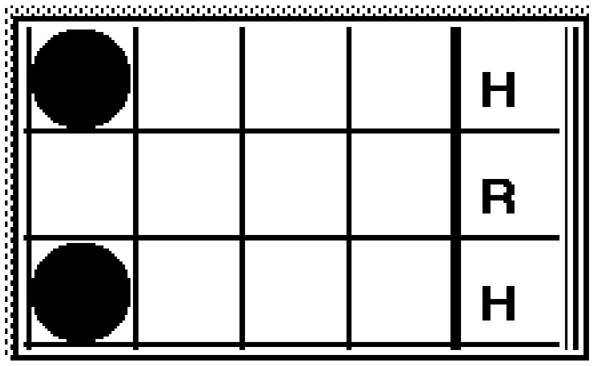
You can use the function FABRICIZE (included in this file) on an arbitrary bitmap to return a fabric-like bitmap (the original bitmap is unaltered).

USAGE: (FABRICIZE BITMAP)

Black Box
A Game






H
R
H

BUTTONS

Johannes A. G. M. Koomen
(Koomen.wbst@Xerox)

Created: November, 1986
Last Edit: December 2, 1988

DESCRIPTION

BUTTONS is a facility for creating icons which will trigger actions when they are clicked in. Each button has a label and an action associated with it. There are three different things which one can do with buttons: Trigger the action, move the button, and bring up a button command menu. These are initiated by use of the left, middle, and right mouse buttons within each button. The command menu is also available through the background menu entry "Button Control." The button world can be tailored (somewhat) using button properties.

Clicking with the Left mouse button on a button and then letting up causes the action associated with the button to be taken. If the action is a list it will be evaluated, otherwise it is stuffed into the system read buffer. The button inverts while the action is being taken.

Clicking with the Middle mouse button allows one to move the button on the screen. The button moves on a grid, unless the left shift key is down.

Clicking with the Right mouse button brings up a menu with the following commands:

Redisplay -- redisplay this button
Move -- same as clicking with the middle button
Copy -- make a copy of this button and move it
Edit -- invokethe structure editor on the label and the action of this button
Close -- close this button (but keep it)
Rollout: **Close All Buttons** -- close all open buttons
Delete -- delete this button
Rollout: **Delete All Buttons** -- delete all existing buttons
Create Button -- make a new button and move it

Expose Buttons -- redisplay all buttons (including previously closed ones)
Align Buttons -- prompts for alignment axis, then for successive buttons to line up
Save Buttons -- save current buttons and button properties in default data file
Rollout: **Save Some Buttons** -- prompt for file and for which buttons to save
Restore Buttons -- discard current buttons, restore saved buttons and properties
Rollout: **Load Some Buttons** -- prompt for file to load, keep/discard current buttons

When BUTTONS is loaded, a single "Create Button" button is placed in the lower left corner of the screen. See RESTORE.BUTTONS below for setting up your buttons programmatically.

FUNCTIONAL INTERFACE

(CREATE-BUTTON *action label location noopenflg*) [Function]

Creates a button with indicated action and label at the given location and displays it unless *noopenflg* is non-NIL. If *action* is NIL, *label* will be used for *action*. If *label* is NIL, (CAR *action*) will be used for *label* if *action* is a list, *action* otherwise. If both *action* and *label* are NIL, the values of the button properties DEFAULT-ACTION and DEFAULT-LABEL will be used instead. If *location* is not a POSITION or a REGION, the user is prompted for a location.

(BUTTONSPROP *propname {newvalue}*) [Function]

Returns the current value of the button property *propname*. If *newvalue* is given, it becomes the new value. The following properties (and their initial values) are currently in use :

DEFAULT-LABEL	"Create Button"
DEFAULT-ACTION	(CREATE-BUTTON)
MENU-FONT	(MODERN 12 BOLD)
LABEL-FONT	(MODERN 10 BOLD)
GRID-ORIGIN	(15 . 15)
SAVE-DIRECTORY	NIL
EDIT-SHADE	4104
EXEC-SHADE	65535

The value of SAVE-DIRECTORY must be acceptable to the function DIRECTORYNAME (i.e., either NIL for login host & directory, or T for current directory, or a standard host & directory spec).

(RESTORE-BUTTONS *filename keep-current-buttons?*) [Function]

Reinstalls the buttons stored on *filename*, which defaults to SAVED-BUTTONS.DATA on the directory indicated by the button property SAVE-DIRECTORY. Existing buttons are discarded unless *keep-current-buttons?* is non-NIL.

(SAVE-BUTTONS *filename buttons*) [Function]

Saves the given *buttons* in *filename*, which defaults to SAVED-BUTTONS.DATA on the directory indicated by the button property SAVE-DIRECTORY. If *buttons* is NIL, all current buttons will be saved.

(SAVE-SOME-BUTTONS *filename buttons*) [Function]

Saves the given *buttons* in *filename*. If *buttons* is NIL, you are prompted to indicated the buttons to be saved. If *filename* is NIL, you are prompted to supply a file in which to save the indicated buttons.

(LOAD-SOME-BUTTONS *filename*) [Function]

Loads the buttons in *filename*. You are prompted to indicate whether to keep the current buttons or discard them. If *filename* is NIL, you are prompted to supply a file from which to load the buttons.

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 tou 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}<LISPFILES>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 efater 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 cround 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.

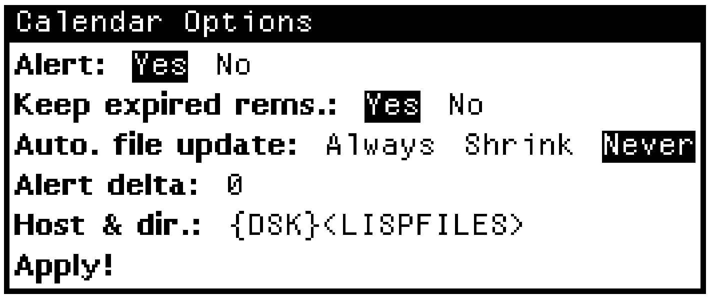
CALTUNE [Variable]

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}<LISPFILES>

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 CALDUPDATEONSHRINKFLG.

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-1988 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 pecularities, 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 ^X-^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 ^X^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 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 GAPTELNET 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 mechnaism 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.

CHECKSET

By: >>Your Name<< (>>Your net address<<)

>>Other packages necessary to run this one<<

This document last edited on >>DATE<<

This package checks source files against compiled files in a directory and prompts you about whether you want to (RE)COMPILE the files that need it. It compares the FILECREATED expressions, and determines whether a BRECOMPILE with CHANGES will suffice or if it is necessary to BCOMPL the file.

(CHECKSET FILES COMPFLG) [Function]

FILES is a list of files. If FILES is NIL, CHECKSET is driven by the variable FILESETS.

COMPFLG can be:

N	don’t compile, just return list. List can be passed to COMPFILES.
Y or NIL	compile
ASK	ask, for each file, whether to compile it.

FILESETS [Variable]

Used by (CHECKSET NIL): FILESETS is a list of variables, each of which has a value that is a list of files. (CHECKSET NIL) peforms (for X in FILESETS join (CHECKSET (EVALV X))). For example,

if FILESETS = (0LISPSET 1LISPSET) and 0LISPSET = (ATERM LLREAD BREAK), 1LISPSET = (WINDOW EDIT) then (CHECKSET) will check if each of those files in turn need recompiling.

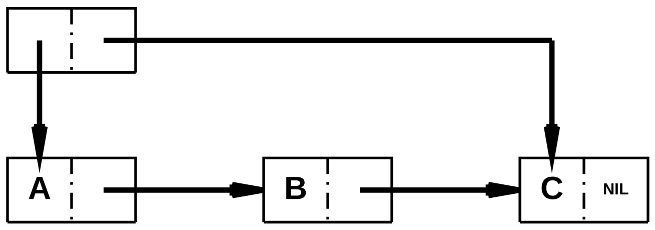
(COMPFILES LST) [Function]

takes a list of elements of the form (RECOMPILE FILE) (COMPILE FILE) as returned by CHECKSET with COMPFLG=N and performs the corresponding operation.

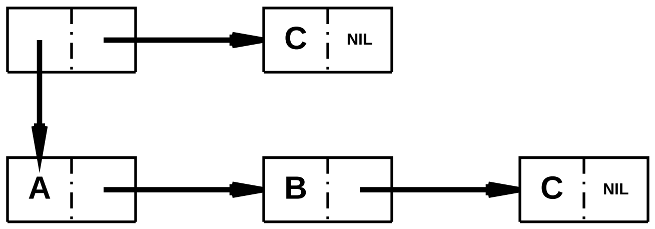
CIRCLPRINT

HPRINT is designed primarily for dumping circular or reentrant list structures (as well as other data structures for which READ is not an inverse of PRINT) so that they can be read back in by Interlisp. The CIRCLPRINT package is designed for printing circular or reentrant structures so that the user can look at them and understand them.

A reentrant list structure is one that contains more than one occurrence of the same EQ structure. For example, TCONC makes use of reentrant list structure so that it does not have to search for the end of the list each time it is called. Thus, if *X* is a list of three elements, (A B C), being constructed by TCONC, the reentrant list structure used by TCONC for this purpose is:

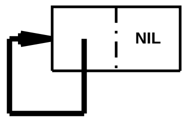


This structure would be printed by PRINT as ((A B C) C). Note that PRINT would produce the same output for the nonreentrant structure:

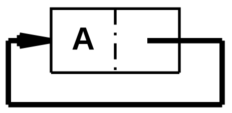


In other words, PRINT does not indicate the fact that portions of the structure in the first figure are identical. Similarly, if PRINT is applied to a circular list structure (a special type of reentrant structure) it will never terminate.

For example, if PRINT is called on the structure:



it will print an endless sequence of left parentheses, and if applied to:



will print a left parenthesis followed by an endless sequence of *A*'s.

The function CIRCLPRINT described below produces output that will exactly describe the structure of any circular or reentrant list structure. This output may be in either single- or double-line format. Below are a few examples of the expressions that CIRCLPRINT would produce to describe the structures discussed above.

First figure, single-line:

((A B *1* C)1)

First figure, double-line:

```
((A B  C)  1)
   1
```

Third figure, single-line:

(*1* 1)

Third figure, double-line:

```
(1)
1
```

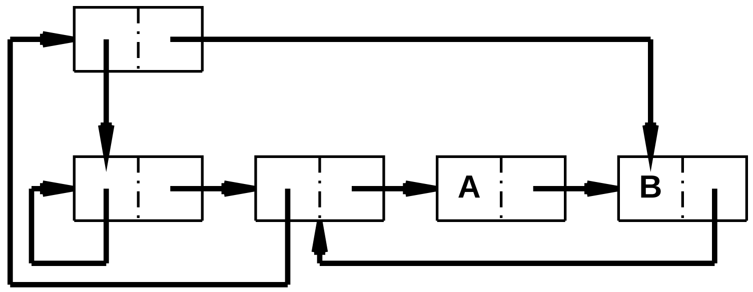
Fourth figure, single-line:

(*1* A . 1)

Fourth figure, double-line:

```
(A . 1)
1
```

The more complex structure:



is printed as follows:

Single-line:

(*2* (*1* 1 *3* 2 A *4* B . 3) . 4)

Double-line:

```
(( 1   2 A  B . 3) . 4)
21    3    4
```

In both formats, the reentrant nodes in the list structure are labeled by numbers. (A reentrant node is one that has two or more pointers coming into it.) In the single-line format, the label is printed between asterisks at the beginning of the node (list or tail) that it identifies. In the double-line format, the label is printed below the beginning of the node it identifies. An occurrence of a reentrant node that has already been identified is indicated by printing its label in brackets.

(CIRCLPRINT *LIST PRINTFLG RLKNT*) [Function]

Prints an expression describing *LIST*. If *PRINTFLG*=NIL, double-line format is used, otherwise single-line format. CIRCLPRINT first calls CIRCLMARK, and then calls either RLPRIN1 (if *PRINTFLG*=T) or RLPRIN2 (if *PRINTFLG*=NIL). Finally, RLRESTORE is called to restore *LIST* to its unmarked state. Returns *LIST.*

(CIRCLMARK *LIST RLKNT*) [Function]

Marks each reentrant node in *LIST* with a unique number, starting at *RLKNT* plus one (or one, if *RLKNT* is NIL). Value is *RLKNT*. Marking *LIST* physically alters it. However, the marking is performed undoably. In addition, *LIST* can always be restored by specifically calling RLRESTORE.

(RLPRIN1 *LIST*) [Function]

Prints an expression describing *LIST* in the single-line format. Does not restore *LIST* to its unCIRCLMARKed state. *LIST* must previously have been CIRCLMARKed, or an error is generated.

(RLPRIN2 *LIST*) [Function]

Same as RLPRIN1, except that the expression describing *LIST* is printed in the double-line format.

(RLRESTORE *LIST*) [Function]

Physically restores list to its original, unmarked state.

Note that the user can mark and print several structures that together share common substructures, e.g., several property lists, by making several calls to CIRCLMARK, followed by calls to RLPRIN1 or RLPRIN2, and finally to RLRESTORE.

(CIRCLMAKER *LIST*) [Function]

LIST may contain labels and references following the convention used by CIRCLPRINT for printing reentrant structures in single-line format, e.g., (*1* . 1). CIRCLMAKER performs the necessary RPLACAs and RPLACDs to make *LIST* correspond to the indicated structure. Value is (altered) *LIST.*

(CIRCLMAKER1 *LIST)* [Function]

Does the work for CIRCLMAKER. Uses free variables LABELST and REFLST. LABELST is a list of dotted pairs of labels and corresponding nodes. REFLST is a list of nodes containing references to labels not yet seen. CIRCLMAKER operates by initializing LABELST and REFLST to NIL, and then calling CIRCLMAKER1. It generates an error if REFLST is not NIL when CIRCLMAKER1 returns. The user can call CIRCLMAKER1 directly to ‘‘connect up’’ several structures that share common substructures, e.g., several property lists.

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 is 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.

COLORDEMO

Maintained By: Frank Shih (Shih.envos@Xerox.com)

Uses: Color, Peano, ColorPolygons

This document last edited on 8-Nov-88

Color Demonstration Programs

The following functions are on file on COLORDEMO.LCOM.

(COLORDEMO) [Function]

brings up a menu of color demonstration programs. The system cycles through the entries on the menu automatically, allowing each to run for a small, fixed amount of time (typically 40 seconds). Selecting one of the entries in the menu causes it to start that program.

(COLORKINETIC *Wait Window*) [Function]

runs a color version of the standard *Kinetic* demo in *Window.* The demo works by BLTSHADEing random textures to *Window* to random regions of *Window* using random OPERATIONs with a bias towards OPERATION='REPLACE.

(VINEDEMO *Wait Window*) [Function]

draws a twisting vine that changes in thickness, direction, and color as it grows inside *Window*.

(RAINING *Wait Window*) [Function]

drops of rain appear to splash on to *Window* causing concentric circular ripples of color to spread outward on the surface of *Window*.

(MODARTDEMO *Wait Window*) [Function]

some of the art produced by this demo is at least as good as some that you will see in art galleries. The demo actually works by BITBLTing *Window* on to itself with a displacement with random SOURCETYPE and OPERATION, mixed in with some BITBLTed random textures.

(STARBURSTDEMO *Wait Window*) [Function]

far far away in a galaxy somewhere in the future, an unexplained physical force sweeps over peaceful stars turning them at once into brilliant exploding novas which are safely viewed at a distance through the rear view porthole of our fleeing spaceship.

(COLORPEANODEMO *Wait Window*) [Function] the Peano fractal curve in color.

(BUBBLEDEMO *Wait Window*) [Function]

the *Window* fills with brilliantly colored soap bubbles. This demo works by calling FILLCIRCLE.

(OVERPAINTDEMO *Wait*) [Function]

uses masking techniques to print over the lower right color demo window. Notice that just the pixels of the character images get printed and not the white pixels that normally surround the character images.

(TILEDEMO *Wait*) [Function]

takes what currently appears in the four color demo windows and adds their images to a growing list called TILEBITMAPS. The demo then tiles the color screen background followed by repeatedly tileing the four color demo windows with randomly chosen tiles.

(TUNNEL *Speed*) [Function]

draws a series of concentric rectangles of increasing size in increasing color numbers. *Speed* determines the size of the rectangles. This can then be ‘‘run’’ by calling ROTATEIT, which is described below.

(MINESHAFT *N OutFlg*) [Function]

draws a series of concentric rectangles of size *N* in increasing color numbers. *OutFlg* determines whether the color numbers increase or decrease. This can then be ‘‘run’’ by calling ROTATEIT, which is described below.

(WELLDEMO *Wait*) [Function]

draws a series of concentric circles on the color demo windows in increasing color numbers. The circles are then "run" by rotating the color map.

(ROTATEIT *BeginColor EndColor Wait*) [Function]

goes into an infinite loop rotating the screen color map. The colors between *BeginColor* (default zero) and *EndColor* (default maximum color) are rotated. If *Wait* is given, (DISMISS *Wait*) is called each time the color map is changed. This provides an easy way of ‘‘animating’’ screen images.

(COLORPOLYDEMO *ColorStream*) [Function]

is on the file COLORPOLYGONS.DCOM. It runs a version of the *Polygons* program on the color screen.

COLORNNCC

By: >>Your Name<< (>>Your net address<<)

>>Other packages necessary to run this one<<

This document last edited on >>DATE<<

INTRODUCTION

This package is the Xerox Lisp software driver for Number Nine Computer Corporation's Revolution 512 x 8 color card. The IJCAI 1985 show featured an Interlisp-D color window system running on a Xerox 1108 attached to a Busmaster and color display. That color window system was based partly on a version of this package. This advanced high level software has become available with the Lyric release or Xerox Lisp.

NECESSARY HARDWARE

You need a Xerox 1108 with Extended Processor Option (CPE), Xerox Busmaster card, an IBM PC expansion chasis, a Number Nine Computer Corporation Revolution 512 x 8 color card, and a third party color display. Please contact your Xerox representative for details concerning acquiring and setting up all the required hardware.

Assuming you have all the hardware you need, turn it all on. This means

(1) Your 1108 is running Xerox LISP.

(2) Your PC expansion chassis is plugged in and powered on.

(3) A cable connects between your 1108 CPE board and your Busmaster board. (The Busmaster board does not go into the 1108, but should rest outside the 1108.)

(4) Another cable connects between your PC expansion chassis and your Busmaster board.

(5) A pair of purple and orange wires connects your Busmaster board to the +5V/Gnd power supply terminals on the side of your 1108.

(6) Your Number Nine Revolution 512 x 8 board is plugged into the PC expansion chassis.

(7) Your color display is plugged in and powered on.

(8) Three cables for red, green, and blue signal connect your Number Nine card to your color display.

Any reconnections that involve (3), (4), or (5) should be made while your 1108 is off. Until you issue some software commands, a black display is normal.

Check that your hardware is set up correctly by typing

(\COLORNNCC.STARTBOARD) [Function]

Taking less than a second to execute, there should be a noticable flicker on your color display, followed by what can be taken to be a stable abstract pattern representing the contents of the Number Nine card’s RAM when the PC chassis was turned on. If \COLORNNCC.STARTBOARD doesn’t return and simply waits, there could be something wrong with the BusMaster card. If \COLORNNCC.STARTBOARD does return, but the image is noisy instead of stable, it could be that you have to adjust the frequency selector on your color display.

COLORNNCC SOFTWARE

The COLORNNCC package provides the machine dependent portion of software that is needed to drive your color display assuming you are using an 1108 with COLORNNCC card and Busmaster. Other than LOADing the COLORNNCC package and turning the COLORNNCC package on using the function COLORDISPLAY, all additional functionality is provided by and documented with the COLOR package. There are no COLORNNCC functions that the user needs to call directly. The user calls functions described in the COLOR documentation.

Once your hardware is on, you can proceed to issue COLOR commands to your hardware. You should have the COLORNNCC package already LOADed from your LIBRARY directory. That is, you’ve already done something like (LOAD ’<LIBRARY>COLORNNCC.DCOM). At this point it may be convenient to follow this documentation along with the documentation for COLOR in the Lisp Library Packages Manual. If you now type

```
(COLORDISPLAY ’ON ’REV512X8)
```

your display will now change from total black to a color test pattern with horizontal and vertical stripes. The sequence of events is that there should be a noticable flicker on your color display, followed by what can be taken to be an abstract pattern representing the contents of the Number Nine card’s RAM when the PC chassis was turned on, followed by a white wall covering up this abstract pattern, followed by the painting of this white wall with horizontal and vertical strpes of color woven together. There are now some simple tests you can do to satisfy yourself that your hardware is working. Here is a small list of things to try:

```
    (SETQ CSBM (COLORSCREENBITMAP))
    (BLTSHADE ’WHITE CSBM)
    (BLTSHADE ’RED CSBM)
    (BLTSHADE ’GREEN CSBM)
    (BLTSHADE ’BLUE CSBM)
    (SETQ DS (DSPCREATE CSBM))
    (DRAWLINE 0 0 500 500 10 ’REPLACE DS ’YELLOW)
    (DRAWLINE 500 0 0 500 10 ’REPLACE DS ’CYAN)
```

Assuming all has gone well to this point, you should now be able to try all the functions described in the COLOR package documentation. The COLORDEMO package is a good source of test programs to try — (IL:LOAD 'COLORDEMO.LCOM) to get this package. Both COLOR and COLORDEMO documentation are in your LispUsers' Manual.

COMMWINDOW

By: Larry Masinter, Stan Lanning, Nick Briggs, Richard Burton (Masinter.pa@Xerox.com, Lanning.PA@Xerox.COM, Briggs.PA@Xerox.com, Burton.pa@Xerox.com)

Uses: COURIERSERVE

This document last edited on 2-Apr-87 17:49:35

INTRODUCTION

This is a demonstration of communication capabilities. The COMMWINDOW module implements a "remote window" capability, where one user can watch (with slow update) a region of another users screen.

Both participants need to have COMMWINDOW loaded.

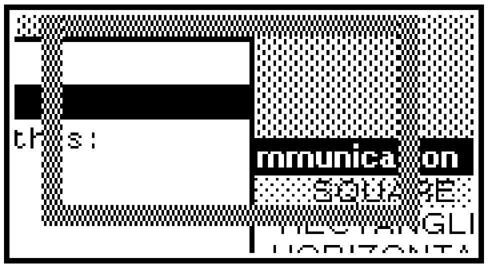
To show someone else a piece of your screen, call

(SEND-BITS *partner &OPTIONAL frame*) [Function]

The partner is the name of the machine you want to talk to. (The watcher has to be registered in the clearinghouse database; this is a NS protocol name.)

frame, if supplied, is a screen region to show. If it is omitted, send-bits will prompt the (sender) for a frame of the screen.

The sender has complete control over the frame. The frame appears as a gray frame around the area shown, like this:



(The frame consists really of 4 thin windows, so that you can really type/button inside it.)

The frame can be moved by left buttoning anywhere on it. Right buttoning on it makes it go away temporarily.

(Reshaping the frame can be accomplished by left buttoning the frame while the shift key is depressed. This doesn't reshape the remote window currently, so this isn't very useful.)

Since this is an experiment, there are a couple of parameters you can vary. The function (mode-menu) will bring up a window that lets you control any and all of these parameters.

The first is the shape of the areas sent in each packet.

Currently available shapes are:

square:	a square of bits as big as will fit in a single packet
rectangle:	4 times wider than it is high.
horizontal:	as wide as the frame, and then as high can be
vertical:	as high as the frame and as wide as can be
h3:	this is a funny mode, where the update is in horizontal chunks, but spread out.

PARAMETERS

The parameter \ETHERLIGHTNING lets you tell the low-level ethernet code to randomly drop packets. This parameter can be used to study the effects of noisy communication lines on transmision protocol.

Another parameter gives you some local control over the region within the frame that is updated. There are three available values:

Sender:	Update near the sender's cursor, when it has moved, ASAP.
Viewer:	Update near the viewer's cursor, when it has moved, ASAP.
NIL:	don't do anything special.

Another parameter lets lets you prune out sending packets that update regions of the display that have not changed. The tradeoff here between the cost of sending the packets vs testing a region to see if it has changed.

A recent addition was the following part of the protocol: when the mouse on the sender is moving, it will send a square around the mouse interleaved with any other transmission ongoing. That means you can make sure a remote area is being updated by wiggling the mouse.

The sender's mouse is also tracked in the remote viewers viewpoint (the cursor shape is currently not tracked, however). The mouse coordinates are sent every packet, so that mouse position always updates.

The viewer has a limited option to place a pointer back on the sender's screen: if the viewer holds the shift key down while the viewer's mouse is in the view point window, a little pointer () will appear in the senders window and track the viewers cursor. (The pointer is in fact a little icon window.) When the viewer releases the shift key, the pointer icon disappears.

CAUTIONS:

Don't move the frame off the screen. It will just signal an error.

The "don't change unsent tiles" parameter doesn't seem to work.

COMPAREDIRECTORIES

By: Larry Masinter and Ron Kaplan

This document edited on

December 2, 1987

December 28, 1998 (Ron Kaplan)

April 7, 2018 (Ron Kaplan)

Rewritten August 25, 2020 (Ron Kaplan)

COMPAREDIRECTORIES compares the contents of two directories, identifying files according to their creation dates and lengths. It is called using the function

(COMPAREDIRECTORIES *DIR1 DIR2 SELECT FILEPATTERNS EXTENSIONSTOAVOID USEDIRECTORYDATES OUTPUTFILE ALLVERSION*S) [Function]

Compares the creation dates of files with matching names in the lists that CDFILES returns for DIR1 and DIR2. Collects or prints CDENTRIES for those files that meet the SELECT criteria. May also collect or print entries for relevant files that exist in DIR1 or DIR2 but not both.

SELECT specifies which the match/mismatch criteria for filtering the output. If SELECT is or contains

AFTER or >: select entries where file1 has a later date than file2

BEFORE or <: select entries where file1 has an earlier date than file2

SAMEDATE or =: select entries where file1 and file2 have the same date

-*: exclude entries where file1 does not exist

*-: exclude entries where file2 does not exist

~=: exclude entries where file1 and file2 are byte-equivalent

SELECT = NIL is equivalent to (< > -* *-). Excludes files with matching dates, a useful default for identifying files that may require further attention.

SELECT = T is equivalent to (= < > -* *-). Includes all files for processing by other functions or later filtering by CDSUBSET (below).

SELECT may also contain the token AUTHOR to indicate that authors should be provided in the printed output (see CDPRINT below).

Unless USEDIRECTORYDATES, the FILECREATED date is used for the date comparison of Lisp source and compiled files, otherwise the file-system CREATIONDATE is used.

If OUTPUTFILE=NIL, then a list of the form (Parameters . entries) is returned. Parameters is a list (DIR1 DIR2 SELECT DATE) that records the parameters of the comparison. Entries contains one entry for each of the file-comparisons that meets the SELECT criteria. Each entry is a CDENTRY record with fields

(matchname info1 daterel info2 equiv)

where matchname is the name.extension shared by the two files, and each file info is either NIL (for nonexistent files) or a CDINFO record with fields

(FULLNAME DATE LENGTH AUTHOR TYPE EOL)

TYPE is SOURCE for Lisp source (filecreated) files, COMPILED for Lisp compiled files, otherwise the PRINTFILETYPE (TEXT, TEDIT...) or NIL. EOL is CR, LF, CRLF, or NIL.

When both files exist, the date relation is one of <, =, or >. Otherwise, the date relation is * if only one file exists. EQUIV is EQUIVALENT for files with different dates but exactly the same bytes, otherwise NIL.

If OUTPUTFILE is not NIL, then it is a filename or open stream on which selected entries will be printed (T for the terminal) by CDPRINT.

COMPAREDIRECTORIES sets the variable LASTCDENTRIES is set to the selected entries. This is used by the functions below if their CDENTRIES is NIL.

(CDFILES DIR FILEPATTERNS EXTENSIONSTOAVOID ALLVERSIONS DEPTH) [Function]

Returns a list of full filenames for files in directory DIR (NIL=T=the connected directory) that match the other file-name filtering criteria. Files are excluded if:

Their name does not match a pattern in FILEPATTERNS (NIL = *). Dotted files are excluded unless FILEPATTERNS includes .* and files in subdirectories are excluded if the number of subdirectories exceeds DEPTH (below).

Their extension is in the list EXTENSIONSTOAVOID (* excludes all extensions).

They are not the highest version unless ALLVERSIONS=T.

DEPTH controls the depth of subdirectory exploration. T means all levels, NIL means no subdirectories. Otherwise the maximum number of ">" characters below the starting DIR in the fullname of files.

(CDFILES) produces all the newest, undotted files in the immediate connected directory.

(CDPRINT CDENTRIES FILE PRINTAUTHOR) [Function]

Prints CDENTRIES on FILE, with one line for each entry. The line for each entry is of the form

FILE1 (AUTHOR) SIZE DATE relation DATE FILE2 (AUTHOR) SIZE

For example

```
ACE.;1 (Joe) 4035 2-May-1985 18:03:54 < 30-Sep-1985 11:14:48 ACE.;3 (Sam) 5096.
```

The line for byte-equivalent files is prefixed with ==. If the files are equivalent except for a difference in end-of-line conventions, the equivalence prefix will indicate the convention for each file (C for CR, L, for LF, 2 for CRLF). Thus C2 indicates that the files are equivalent except that file1 marks line ends with CR and file2 with CRLF.

Note that because of the setting of LASTCDENTRIES, evaluating (CDPRINT) after COMPAREDIRECTORIES prints the results of the last comparision.

For conciseness, authors are included only if PRINTAUTHOR or if AUTHOR is included in the SELECT parameter of CDENTRIES. Also, the redundant file-name hosts/directories are not printed.

(CDMAP CDENTRIES FN) [Function]

(CDSUBSET CDENTRIES FN) [Function]

CDMAP applies FN to each CDENTRY in CDENTRIES. CDSUBSET applies FN and also returns the subset of CDENTRIES for which FN is non-NIL and preserves in the value the parameters of CDENTRIES. For convenience, at each invocation the variables MATCHNAME INFO1 DATEREL INFO2 and EQUIV are bound to the corresponding fields and can be used freely by FN.

USEFUL UTILITIES

(FIX-DIRECTORY-DATES FILES) [Function]

For every file included in or specified by FILES, if it is a Lisp source or compiled whose directory creation date is more than 30 seconds later than its internal filecreated date (presumably because of copying), then its directory date is reset to match the internal date. FILES can be a list of file names or a pattern interpretable by FILDIR. Returns a list of files whose dates have been changed.

(FIX-EQUIV-DATES CDENTRIES) [Function]

If there is an entry in CDENTRIES whose files are EQUIVALENT but with different directory creation dates, the directory date of the file with the later date (presumably a copy) is reset to match the date of the earlier file. In the end all equivalent files will have the same (earliest) date. Returns a list of files whose dates have been changed.

(COPY-MISSING-FILES CDENTRIES TARGET MATCHNAMES) [Function]

Target is 1 or 2, indicating the direction of potential copies. If an entry with a source file but no target file has a matchname in MATCHNAMES, the source file is copied to the target directory. All target-absent files are copied if MATCNAMES is NIL. Source properties (including version number) are preserved in the target.

(COPY-COMPARED-FILES CDENTRIES TARGET MATCHNAMES) [Function]

Target is 1 or 2, indicating the direction of potential copies. If an entry with both source and target files has a matchname in MATCHNAMES, the source file is copied to a new version of the target file. All files are copied if MATCHNAMES is NIL.

`(COMPARE-ENTRY-SOURCE-FILES CDENTRY LISTSTREAM EXAMINE DW?)`

[Function]

This is a simple wrapper for calling COMPARESOURCES if the CDENTRY files are Lisp source files. The function (CDENTRY MATCHNAME CDENTRIES is useful for extracting a particular entry, with CDENTRIES defaulting to LASTCDENTRIES.

(COMPILED-ON-SAME-SOURCE CDENTRIES) [Function]

Returns the subset of entries with Lisp compiled files (dfasl or lcom) that are compiled on the same source, according to SOURCE-FOR-COMPILED-P below. Presumably one should be removed to avoid confusion.

(FIND-SOURCE-FILES CFILES SDIRS DFASLMARGIN) [Function]

Returns (CFILE . SFILES) pairs where CFILE is a Lisp compiled file in CFILES and SFILES is list of files in SDIRS that CFILE was compiled on according to SOURCE-FOR-COMPILED-P. This suggests that at least one of SFILES should be copied to CFILE's location (or vice versa).

(FIND-COMPILED-FILES SFILES CDIRS DFASLMARGIN) [Function]

Returns (SFILE . CFILES) pairs where SFILE is a Lisp source file in SFILES and CFILES are files in CDIRS that are compiled on SFILE according to SOURCE-FOR-COMPILED-P. This suggests that at least one of CFILES should be copied to SFILE's location.

(FIND-UNCOMPILED-FILES FILES DFASLMARGIN COMPILEXTS) [Function]

Returns a list of elements each of which corresponds to a source file in FILES for which no appropriate compiled file can be found. An appropriate compiled file is a file in the same location with extension in COMPILEEXTS (defaulting to *COMPILED-EXTENSIONS*) that satisfies SOURCE-FOR-COMPILED-P. Each element is a list of the form

(sourcefile . cfiles)

cfiles contains compiled files that were compiled on a different version of sourcefile, NIL if no such files exist. Each cfile item is a pair (cfile timediff) where timediff is the time difference (in minutes) between the creation date of the compiled-file's source and the creation date of sourcefile (positive if the cfile was compiled later, as should be the case). FILES can be an explicit list of files, or a file specification interpretable by FILDIR; in that case only the newest source-file versions are processed.

(FIND-UNSOURCED-FILES CFILES DFASLMARGIN COMPILEXTS) [Function]

Returns the subset of the compiled files specified by CFILES for which a corresponding source file according to SOURCE-FOR-COMPILED-P cannot be found in the same directory. CFILES can be a list of files or a pattern that FILDIR can interpret. COMPILEEXTS can be one or more explicit compile-file extensions, defaulting to *COMPILED-EXTENSIONS*.

(SOURCE-FOR-COMPILED-P SOURCE COMPILED DFASLMARGIN) [Function]

Returns T if it can confirm that Lisp COMPILED file was compiled on Lisp SOURCE file. SOURCE and COMPILED can be provided as CREATED-AS values, to avoid repetitive computation. This compares the information in the filecreated expressions, original file names and original dates, and not the current directory names and dates.

It appears that the times in DFASL files may differ from the filecreated source dates by a few minutes. The DFASLMARGIN can be provided to loosen up the date matching criterion. DFASLMARGIN is a pair (max min) and a DFASL COMPILED is deemed to be compiled on SOURCE if the compiled's source date is no more than max and no less than min minutes after the source date. A negative min allows for the possibility that the compiled-source date is earlier than the candidate source date. DFASLMARGIN defaults to (20 0). A single positive number x is coerced to (x 0). A single negative number is coerced to (-x x) (compiled file is no more than x minutes later or earlier). T is infinity in either direction. Examples:

(T 0): COMPILED compiled on source later than SOURCE
(0 T): COMPILED compiled on source earlier than SOURCE (odd)
12: COMPILED compiled on source later than SOURCE by no more than 12 minutes
-12: COMPILED compiled on source 12 minutes before or after SOURCE

(FIND-MULTICOMPILED-FILES FILES SHOWINFO) [Function]

Returns a list of files in FILES that have more than one type of compiled file (e.g. LCOM and DFASL). FILES is interpretable by FILDIR. If SHOWINFO, then the value contains a list for each file of the form

(rootname loaded-version . CREATED-AS information for each compile-type)

Otherwise just the rootname of the source is returns.

(CREATED-AS FILE) [Function]

If FILE is a Lisp source or compiled file, returns a record of its original filename and filecreated dates, and for compiled files, also the original compiled-on name and date. The return for a source file is a pair

(sfullname sfilecreateddate)

The return for a compiled file is a quadruple

(cfullname cfilecreated sfullname sfilecreateddate)

where sfullname and sourcefilecreated are extracted from the file’s compiled-on information. The return is (fullname NIL) for a non-Lisp file.

(EOLTYPE FILE SHOWCONTEXT) [Function]

Returns the EOLTYPE of FILE (CR, LF, CRLF) if the type is unmistakable: contains at least one instance of one type and no instances of any others. Returns NIL if there is evidence of inconsistent types. If SHOWCONTEXT is an integer, it is the number of bytes for EOLTYPE to display before and after an instance of an inconsistent type. At each instance, the user is asked whether to continue scanning for other instances. SHOWCONTEXT = T is interpreted as 100.

(BINCOMP FILE1 FILE2 EOLDIFFOK) [Function]

Returns T if FILE1 and FILE2 are byte-identical. If EOLDIFFOK and FILE1 and FILE2 differ only in their eol conventions, the value is a list of the form (EOL1 EOL2), e.g. (CR CRLF). Otherwise the value is NIL.

COMPARESOURCES

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

Browser added by Ron Kaplan (12/2021)

INTRODUCTION

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

(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), COMPARESOURCES 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, COMPARESOURCES 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, COMPARESOURCES 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.
2WINDOWS	Call the editor separately for each pair of changed objects.

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 COMPARESOURCES 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 --)".

(CSBROWSER *FILEX FILEY DW? LABEL1 LABEL2*) [Function]

This directs the output of a call to COMPARESOURCESto a scrollable window. Clicking on the output for each identified difference brings up for further examination side-by-side SEDIT windows on the different objects. LABEL1 and LABEL2 are optional strings that override the default way of constructing the title for the broswer window.

FORM OF THE OUTPUT

The output of COMPARESOURCES is in several sections. First, all functions are compared. Then expressions of other types (variables, macros, etc) are compared. When a difference is found, COMPARESOURCES 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, updated by Ron Kaplan 12/2021

Uses TEDIT, GRAPHER, REGIONMANAGER

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 *FILE1 FILE2 HASH.TYPE CHUNKREGION FILELABELS TITLE TEXTWIDTH TEXTHEIGHT*) [Function]

FILE1 and *FILE2* are the names of the two files to compare. The order is not important, except that in the resulting graph the *FILE1* information will appear on the left, and the *FILE2* info on the right. The files may also be provide as input streams.

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.

CHUNKREGION is the region on the display screen used for the file comparison graph, the chunk window. If *CHUNKREGION*=NIL, the system asks the user to specify a region, prompting with a region that is just wide enough for the graph. If *CHUNKREGION*=T, a region in the lower left corner is used. IF *CHUNKREGION* is a position, the chunkwindow will be located relative to that position, with its horizontal midpoint at the specified XCOORD and its top at the YCOORD.

FILELABELS is an optional pair of labels that will appear over the columns of the difference graph instead of the (often overly long) full names of the files.

TITLE is an optional title to be used for the comparison window.

TEXTWIDTH and *TEXTHEIGHT* are optional parameters that control the size of each of the two text-display windows.

COMPARETEXT creates a graph with two columns. Each column contains the label for 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 and its counterpart in the other column to be inverted, and read-only Tedit windows are open on the files with the appropriate text selected. If a Tedit window to a file is already active, the selection is simply moved. If COMPARETEXT.AUTOTEDIT is true (initially), then regions are selected automatically for the Tedit windows, otherwise the mouse must be used to specify ghost regions.

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 chunks (the ones that are boxed), using the hash type selected, and create a new graph window.

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.

If the variable COMPARETEXT.ALLCHUNKS is NIL (initially T), then the graph is abbreviated so that nodes for identical chunks in the same position are not shown.

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 Lispx 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 [Lispx Macro]

The LISPXMACRO 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:

ATOM	Converts to a string on writing and converts to an atom on reading.
FONT	Converts a FONTDESCRIPTOR to a record describing the font on writing and convert the record back to a FONTDESCRIPTOR on reading.
REGION	A sequence of INTEGER.
POSITION	Converts a POSITION record to two integers on writing and converts back to a POSITION record on reading.
NUMBER	Like INTEGER but can also be NIL.
BRUSH	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.
OPERATION	An ENUMERATION of NIL, REPLACE, PAINT, INVERT or ERASE.
TEXTURE	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]

(REMOTEAPPLY *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 SCALEDBITBLT) 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 *COURIERSTREAM PROGRAM PROCEDURE RESULTLST*) [Function]

(COURIER.ABORT *COURIERSTREAM PROGRAM ERROR RESULTLST*) [Function]

(COURIER.REJECT *COURIERSTREAM ERROR RESULTLST*) [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 *STREAM [ABORTFLG]*) [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.

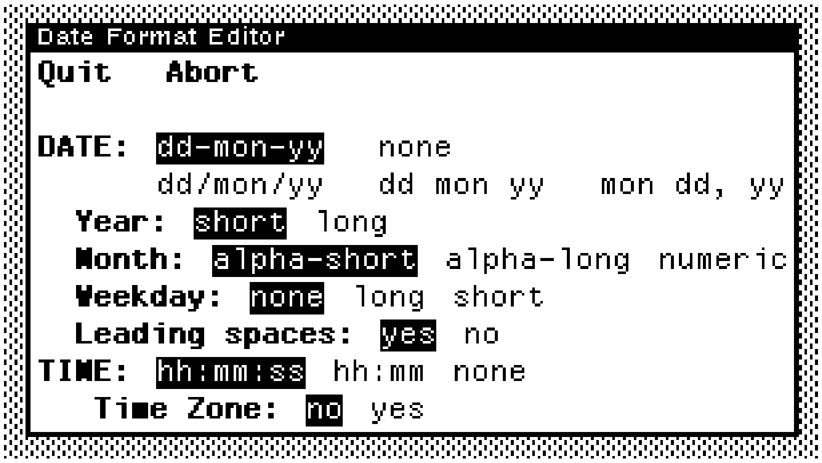
DATEFORMAT-EDITOR

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

This document last edited on May 19, 1989 by Bill van Melle.

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 funciton with argument T if you change the variable DATEFORMAT-EDITOR-ITEMS.

EXTENDED DATEFORMAT OPTIONS

DATEFORMAT-EDITOR supports the following additional DATEFORMAT options by virtue of loading the module DATEPATCH:

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.

DATESORT
[Internal Lafite Utility]

Datesort places an item on the background menu. When selected you will be prompted for the name of a Lafite Mail folder to be sorted, and the name of a new resulting sorted folder. The folder messages will then be sorted in ascending order on their "Date:" fields (this takes some time).

DEBUGGER-CONTEXT

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

This document last edited on August 13, 1987.

Introduction

When debugging Common Lisp programs, have you ever wished that the Xerox Lisp debugger let you do things in the right lexical context? Ever wish you could access locally-defined functions, return from blocks, or evaluate variables in the debugger without resorting to the inspector? Ever wish you could tell the "big boys" in Washington a thing or two? Well, two out of three ain't bad. DEBUGGER-CONTEXT gives you the ability to access and modify the lexical context of code you are debugging in a straightforward way.

User Interface

DEBUGGER-CONTEXT defines two debugger commands that affect the debugger's lexical context.

`lex` [Debugger Command]

Sets the lexical context of the debugger window to that of the stack frame selected in the backtrace window (`il:lastpos`). Henceforth, all evaluation done in this debugger window will be with respect to that lexical environment. The lexical environment of a frame is taken to be the first one encountered as an argument in that frame. Once the `lex` command is given, selecting other stack frames in the backtrace window will not change the lexical environment; you must issue another `lex` command to do so.

The lexical environment of a frame is taken to be the first one encountered as an argument in that frame.

`unlex` [Debugger Command]

Removes the current lexical context; all evaluation will subsequently be done with respect to the current dynamic environment.

DECL

UNSUPPORTED

INTERNAL

Uses: SIMPLIFY, LABEL and LAMBDATRAN

NOTE TO LYRIC/MEDLEY USERS

The DECL module is not supported in Lyric/Medley since it uses the DWIM facilities heavily, and DWIM is not supported in Lyric/Medley. It is being released as a LispUsers module only for backward compatibility. The DECL module only runs under the OLD-INTERLISP-T executive, and all the code that uses it will also have to run under this executive. Therefore, you may wish to convert code that uses DECL to something else.

INTRODUCTION

The Decl LispUsers package is contained on the file DECL.LCOM. The Decl package requires the LambdaTran package. LAMBDATRAN.LCOM will automatically be loaded with Decl if it is not already present.

The Decl package extends Interlisp to allow the user to declare the types of variables and expressions appearing in functions. It provides a convenient way of constraining the behavior of programs when the generality and flexibility of ordinary Interlisp is either unnecessary, confusing, or inefficient.

Decl provides a simple language for declarations, and augments the interpreter and the compiler to guarantee that these declarations are always satisfied. The declarations make programs more readable by indicating the type, and therefore something about the intended usage, of variables and expressions in the code. They facilitate debugging by localizing errors that manifest themselves as type incompatibilities. Finally, the declaration information is available for other purposes: compiler macros can consult the declarations to produce more efficient code; coercions for arguments at user interfaces can be automatically generated; and the declarations will be noticed by the Masterscope function analyzer.

The declarations interpreted by the Decl package are in terms of a set of declaration types called *decltypes*, each of which specifies a set of acceptable values and also (optionally) other type-specific behavior. The Decl package provides a set of facilities for defining decltypes and their relations to each other, including type-valued expressions and a comprehensive treatment of union types.

The following description of the Decl package is divided into three parts. First, the syntactic extensions that permit the concise attachment of declarations to program elements are discussed. Second, the mechanisms by which new decltypes can be defined and manipulated are covered. Finally, some additional capabilities based on the availability of declarations are outlined.

USING DECLARATIONS IN PROGRAMS

Declarations may be attached to the values of arbitrary expressions and to LAMBDA and PROG variables throughout (or for part of) their lexical scope. The declarations are attached using constructs that resemble the ordinary Interlisp LAMBDA, PROG, and PROGN, but which also permit the expression of declarations. The following examples illustrate the use of declarations in programs.

Consider the following definition for the factorial function (FACT *N*):

```
[LAMBDA (N)
   (COND
      ((EQ N 0) 1)
       (T (ITIMES N (FACT (SUB1 N]
```

Obviously, this function presupposes that *N* is a number, and the run-time checks in ITIMES and SUB1 will cause an error if this is not so. For instance, (FACT T) will cause an error and print the message NON-NUMERIC ARG T. By defining FACT as a DLAMBDA, the Decl package analog of LAMBDA, this presupposition can be stated directly in the code:

```
[DLAMBDA ((N NUMBERP))
   (COND
      ((EQ N 0) 1)
       (T (ITIMES N (FACT (SUB1 N]
```

With this definition, (FACT T) will result in a NON-NUMERIC ARG T error when the body of the code is executed. Instead, the NUMBERP declaration will be checked when the function is first entered, and a *declaration fault* will occur. Thus, the message that the user will see will not dwell on the offending value T, but instead give a symbolic indication of what variable and declaration were violated, as follows:

```
DECLARATION NOT SATISFIED
((N NUMBERP) BROKEN):
```

The user is left in a break from which the values of variables, e.g., *N*, can be examined to determine what the problem is.

The function FACT also makes other presuppositions concerning its argument, *N*. For example, FACT will go into an infinite recursive loop if *N* is a number less than zero. Although the user could program an explicit check for this unexpected situation, such coding is tedious and tends to obscure the underlying algorithm. Instead, the requirement that *N* not be negative can be succinctly stated by declaring it to be a subtype of NUMBERP that is restricted to non-negative numbers. This can be done by adding a SATISFIES clause to *N*'s type specification:

```
[DLAMBDA ([N NUMBERP (SATISFIES (NOT (MINUSP N])
   (COND
      ((EQ N 0) 1)
       (T (ITIMES N (FACT (SUB1 N]
```

The predicate in the SATISFIES clause will be evaluated after *N* is bound and found to satisfy NUMBERP, but before the function body is executed. In the event of a declaration fault, the SATISFIES condition will be included in the error message. For example, (FACT -1) would result in:

```
DECLARATION NOT SATISFIED
```

```
((N NUMBERP (SATISFIES (NOT (MINUSP N))) BROKEN):
```

The DLAMBDA construct also permits the type of the value that is returned by the function to be declared by means of the pseudo-variable RETURNS. For example, the following definition specifies that FACT is to return a positive integer:

```
[DLAMBDA ([N NUMBERP (SATISFIES (NOT (MINUSP N]
   [RETURNS FIXP (SATISFIES (IGREATERP VALUE 0])
  (COND
      ((EQ N 0) 1)
       (T (ITIMES N (FACT (SUB1 N]
```

After the function body is evaluated, its value is bound to the variable VALUE and the RETURNS declaration is checked. A declaration fault will occur if the value is not satisfactory. This prevents a bad value from propagating to the caller of FACT, perhaps causing an error far away from the source of the difficulty.

Declaring a variable causes its value to be checked not only when it is first bound, but also whenever that variable is reset by SETQ within the DLAMBDA. In other words, the type-checking machinery will not allow a declared variable to take on an improper value. An iterative version of the factorial function illustrates this feature in the context of a DPROG, the analog of PROG:

```
(DLAMBDA ([N NUMBERP (SATISFIES (NOT (MINUSP N]
   [RETURNS FIXP (SATISFIES (IGREATERP VALUE 0])
[DPROG ([TEMP 1 FIXP (SATISFIES (IGREATERP TEMP 0]
   [RETURNS FIXP (SATISFIES (IGREATERP VALUE 0])
      LP  (COND ((EQ N 0) (RETURN TEMP)))
          (SETQ TEMP (ITIMES N TEMP))
          (SETQ N (SUB1 N))
          (GO LP]
```

DPROG declarations are much like DLAMBDA declarations, except that they also allow an initial value for the variable to be specified. In the above example, TEMP is declared to be a positive integer throughout the computation and N is declared to be non-negative. Thus, a bug which caused an incorrect value to be assigned by one of the SETQ expressions would cause a declaration failure. Note that the RETURNS declaration for a DPROG is also useful in detecting the common bug of omitting an explicit RETURN.

DLAMBDAs

The Decl package version of a LAMBDA expression is an expression beginning with the atom DLAMBDA. Such an expression is a function object that may be used in any context where a LAMBDA expression may be used. It resembles a LAMBDA expression except that it permits declaration expressions in its argument list, as illustrated in the examples given earlier. Each element of the argument list of a DLAMBDA may be a literal atom (as in a conventional LAMBDA) or a list of the form (*NAME TYPE .EXTRAS*). Strictly, this would require a declaration with a SATISFIES clause to take the form (N (NUMBERP (SATISFIES --)) --). However, due to the frequency with which this construction is used, it may be written without the inner set of parentheses, e.g., (N NUMBERP (SATISFIES --) --).

NAME fulfills the standard function of a parameter, i.e., providing a name to which the value of the corresponding argument will be bound.

TYPE is either a Decl package type name or type expression. When the DLAMBDA is entered, its arguments will be evaluated and bound to the corresponding argument names, and then, after all the argument names have been bound, the declarations will be checked. The type checking is delayed so that SATISFIES predicates can include references to other variables bound by the same DLAMBDA. For example, one might wish to define a function whose two arguments are not only both required to be of some given type, but are also required to satisfy some relationship (e.g., that one is less than the other).

EXTRAS allows some additional properties to be attached to a variable. One such property is the accessibility of *NAME* outside the current lexical scope. Accessibility specifications include the atoms LOCAL or SPECIAL, which indicate that this variable is to be compiled so that it is either a LOCALVAR or a SPECVAR, respectively. This is illustrated by the following example:

```
[DLAMBDA ((A LISTP SPECIAL)

          (B FIXP LOCAL))

          ...]
```

A more informative equivalent to the SPECIAL key word is the USEDIN form, the tail of which can be a list of the other functions that are expected to have access to the variable.[1]

```
[DLAMBDA ((A LISTP (USEDIN FOO FIE))

       (B FIXP LOCAL))

       ...]
```

EXTRAS may also include a comment in standard format, so that descriptive information may be given where a variable is bound:

```
[DLAMBDA ((A LISTP (USEDIN FOO FIE)   (* This is an important variable))

          (B FIXP LOCAL))

          ...]
```

As mentioned earlier, the value returned by a DLAMBDA can also be declared, by means of the pseudo-variable RETURNS. The RETURNS declaration is just like other DLAMBDA declarations, except (1) in any SATISFIES predicate, the value of the function is referred to by the distinguished name VALUE; and (2) it makes no sense to declare the return value to be LOCAL or SPECIAL.

DPROG

Just as DLAMBDA resembles LAMBDA, DPROG is analogous to PROG. As for an ordinary PROG, a variable binding may be specified as an atom or a list including an initial value form. However, a DPROG binding also allows *TYPE* and *EXTRAS* information to appear following the initial value form. The format for these augmented variable bindings is (*NAME INITIALVALUE TYPE .EXTRAS*).

The only difference between a DPROG binding and a DLAMBDA binding is that the second position is interpreted as the initial value for the variable. Note that if the user wishes to supply a type declaration for a variable, an initial value *must* be specified. The same rules apply for the interpretation of the type information for DPROGs as for DLAMBDAs, and the same set of optional *EXTRA*s can be used. DPROGs may also declare the type of the value they return, by specifying the pseudo-variable RETURNS.

Just as for a DLAMBDA, type tests in a DPROG are not asserted until after all the variables have been bound, thus permitting predicates to refer to other variables being bound by this DPROG. If NIL appears as the initial value for a binding (i.e., the atom NIL actually appears in the code, not simply an expression that evaluates to NIL) the initial type test will be suppressed, but subsequent type tests, e.g., following a SETQ, will still be performed.

A common construct in Lisp is to bind and initialize a PROG variable to the value of a complicated expression in order to avoid recomputing it, and then to use this value in initializing other PROG variables, e.g.

```
[PROG ((A EXPRESSION))

      (RETURN (PROG ((B... ( A...))

                     (C... ( A... )))

                    ...]
```

The ugliness of such constructions in conventional Lisp often tempts the programmer to loosen the scoping relationships of the variables by binding them all at a single level and using SETQ's in the body of the PROG to establish the initial values for variables that depend on the initial values of other variables, e.g.,

```
[PROG ((A  EXPRESSION) B C)

      (SETQ B (...A... ))

      (SETQ C ( ...A... ))

      ...]
```

In the Decl package environment, this procedure undermines the protection offered by the type mechanism by encouraging the use of uninitialized variables. Therefore, the DPROG offers a syntactic form to encourage more virtuous initialization of its variables. A DPROG variable list may be segmented by occurrences of the special atom THEN, which causes the binding of its variables in stages, so that the bindings made in earlier stages can be used in later ones, e.g.,

```
[DPROG ((A (LENGTH FOO) FIXP LOCAL)

        THEN (B (SQRT A) FLOATP)

        THEN (C (CONS A B) LISTP))

        ...]
```

Each stage is carried out as a conventional set of DPROG bindings (i.e., simultaneously, followed by the appropriate type testing). This layering of the bindings permits one to gradually descend into a inner scope, binding the local names in a very structured and clean fashion, with initial values type-checked as soon as possible.

DECLARATIONS IN ITERATIVE STATEMENTS

The CLISP iterative statement provides a very useful facility for specifying a variety of PROGs that follow certain widely used formats. The Decl package allows declarations to be made for the scope of an iterative statement via the DECLARE CLISP (I.S. operator). DECLARE can appear as an operator anywhere in an iterative statement, followed by a list of declarations, for example:

(for J from 1 to 10 declare (J FIXP) do. . .

Note that DECLARE declarations do not create bindings, but merely provide declarations for existing bindings. For this reason, an initial value cannot be specified and the form of the declaration is the same as that of DLAMBDAs, namely create (*NAME TYPE . EXTRAS*).

Note that variables bound *outside* of the scope of the iterative statement, i.e., a variable used freely in the I.S., can also be declared using this construction. Such a declaration will only be in effect for the scope of the iterative statement.

DECLARING A VARIABLE FOR A RESTRICTED LEXICAL SCOPE

The Decl package also permits declaring the type of a variable over some restricted portion of its existence. For example, suppose the variable X is either a fixed or floating number, and a program branches to treat the two cases separately. On one path X is known to be fixed, whereas on the other it is known to be floating. The Decl package DPROGN construct can be used in such cases to state the type of the variable along each path. DPROGN is exactly like PROGN, except that the second element of the form is interpreted as a list of DLAMBDA format declarations. These declarations are added to any existing declarations in the containing scope, and the composite declaration (created using the ALLOF type expression), is considered to hold throughout the lexical scope created by the DPROGN. Thus, our example becomes:

```
(if (FIXP X)

then (DPROGN ((X FIXP))...else (DPROGN ((X FLOATP)) ...))
```

Like DPROG and DLAMBDA, the value of a DPROGN may also be declared, using the pseudo-variable RETURNS.

DPROGN may be used not only to restrict the declarations of local variables, but also to declare variables that are being used freely. For example, if the variable A is used freely inside a function but is known to be FIXP, this fact could be noted by enclosing the body of the function in (DPROGN ((A FIXP FREE)) *BODY*). Instead of FREE, the more specific construction (BOUNDIN *FUNCTION1 FUNCTION 2*. . .) can be used. This not only states that the variable is used freely but also gives the names of the functions that might have provided this binding.[2]

Since the DPROGN form introduces another level of parenthesization, which results in the enclosed forms being prettyprinted indented, the Decl package also permits such declarations to be attached to their enclosing DLAMBDA or DPROG scopes by placing a DEC expression, e.g., (DECL (A FIXP (BOUNDIN FUM)), before the first executable form in that scope. Like DPROGN's, DECL declarations use DLAMBDA format.

DECLARING THE VALUES OF EXPRESSIONS

The Decl package allows the value of an arbitrary form to be declared with the Decl construct THE. A THE expression is of the form (THE *TYPE . FORMS*), e.g., (THE FIXP (FOO X)). FORMS are evaluated in order, and the value of the *last* one is checked to see if it satisfies *TYPE*, a type name or type expression. If so, its value is returned, otherwise a declaration fault occurs.

ASSERTIONS

The Decl package also allows for checking that an arbitrary predicate holds at a particular point in a program's execution, e.g., a condition that must hold at function entry but not throughout its execution. Such predicates can be checked using an expression of the form (ASSERT *FORM1 FORM2*), in which each *FORM1* is either a list (which will be evaluated) or a variable (whose declaration will be checked). Unless all elements of the ASSERT form are satisfied, a declaration fault will take place.

ASSERTing a variable provides a convenient way of verifying that the value of the variable has not been improperly changed by a lower function. Although a similar effect could be achieved for predicates by explicit checks of the form (OR *PREDICATE* (SHOULDNT)), ASSERT also provides the ability both to check that a variable's declaration is currently satisfied and to remove its checks at compile time without source code modification (see COMPILEIGNOREDECL).

USING TYPE EXPRESSIONS AS PREDICATES

The Decl package extends the Record package TYPE? construct so that it accepts decltypes, as well as record names, e.g., (TYPE? (FIXP (SATISFIES (ILESSP VALUE 0))) EXPR). Thus, a TYPE? expression is exactly the same as a THE expression except that, rather than causing a declaration fault, TYPE? is a predicate that determines whether or not the value satisfies the given type.

ENFORCEMENT

The Decl package is a ‘‘soft’’ typing system—that is, the data objects themselves are not inherently typed. Consequently, declarations can only be enforced within the lexical scope in which the declaration takes place, and then only in certain contexts. In general, changes to a variable’s value such as those resulting from side effects to embedded structure (e.g., RPLACA, SETN, etc.) or free variable references from outside the scope of the declaration cannot be, and therefore are not, enforced.

Declarations *are* enforced, i.e., checked, in three different situations: when a declared variable is bound to some value or rebound with SETQ or SETQQ, when a declared expression is evaluated, and when an ASSERT expression is evaluated. In a binding context, the type check takes place *after* the binding, including any user-defined behavior specified by the type’s binding function. Any failure of the declarations causes a break to occur and an informative message to be printed. In that break, the name to which the declaration is attached (or *VALUE* if no name is available) will be bound to the offending value. Thus, in the FACT T example above, N would be bound to T. The problem can be repaired either by returning an acceptable value from the break via the RETURN command, or by assigning an acceptable value to the offending name and returning from the break via an OK or GO command. The unsatisfied declaration will be reasserted when the computation is continued, so an unacceptable value will be detected.[3]

The automatic enforcement of type declarations is a very flexible and powerful aid to program development. It does, however, exact a considerable run-time cost because of all the checking involved. Factors of two to ten in running speed are not uncommon, especially where low-level, frequently used functions employ type declarations. As a result, it is usually desirable to remove the declaration enforcement code when the system is believed to be bug-free and performance becomes more central. This can be done with the variable COMPILEIGNOREDECL.

COMPILEIGNOREDECL [Variable]

Setting the value of the variable COMPILEIGNOREDECL to T (initially NIL) instructs the compiler not to insert declaration enforcement tests in the compiled code. More selective removal can be achieved by setting COMPILEIGNOREDECL to a list of function names. Any function whose name is found on this list is compiled without declaration enforcement.

IGNOREDECL. *VAL* [File Com]

Declaration enforcement may be suppressed selectively by a file using the IGNOREDECL file package command. If this appears in a file’s file commands, it redefines the value of COMPILEIGNOREDECL to VAL for the compilation of this file only.

Note: The period in the IGNOREDECL file package command is significant. To set COMPILEIGNOREDECL to T, use (IGNOREDECL . T), not (IGNOREDECL T).

DECLTYPES

A Decl package type, or decltype, specifies a subset of data values to which values of this type are restricted. For example, a ‘‘positive number’’ type might be defined to include only those values that are numbers and greater than zero. A type may also specify how certain operations, such as assignment or binding (see BINDFN), are to be performed on variables declared to be of this type.

The inclusion relations among the sets of values that satisfy the different types define a natural partial ordering on types, bound by the universal type ANY (which all values satisfy) and the empty type

NONE (which no value satisfies). Each type has one or more *supertypes* (each type has at least ANY as a supertype) and one or more subtypes (each type has at least NONE as a subtype). This structure is important to the user of Decl as it provides the framework in which new types are defined. Typically, much of the definition of a new type is defaulted, rather than specified explicitly. The definition will be completed by inheriting attributes which are shared by all its immediate supertypes.

An initial set of decltypes that defines the Interlisp built-in data types and a few other commonly used types is provided. Thereafter, new decltypes are created in terms of existing ones using the type expressions described below. For conciseness, such new types can be associated with literal atoms using the function DECLTYPE.

PREDEFINED TYPES

Some commonly used types, such as the Interlisp built-in data types, are already defined when the Decl package is loaded. These types, indented to show subtype-supertype relations, are:[4]

```
ANY
  ATOM            LST
     ARRAYP      STRINGP    FUNCTION    STACKP
    LITATOM         ALIST    HARRAYP
      NIL           LISTP       READTABLEP
    NUMBERP
      FIXP
        LARGEP
        SMALLP
      FLOATP


                                NONE
```

Note that the definition of LST causes NIL to have multiple supertypes, i.e., LITATOM and LST, reflecting the duality of NIL as an atom and a (degenerate) list.

In addition, declarations made using the Record package also define types that are attached as subtypes to an appropriate existing type (e.g., a TYPERECORD declaration defines a subtype of LISTP, a DATATYPE declaration a subtype of ANY, etc.) and may be used directly in declaration contexts.

TYPE EXPRESSIONS

Type expressions provide convenient ways for defining new types in terms of modifications to, or compositions of one or more existing types.

(MEMQ *VALUE1. . .VALUE N*) [Type Expression]

Specifies a type whose values can be any one of the fixed set of elements VALUE 1. . .VALUE N. For example, the status of a device might be represented by a datum restricted to the values BUSY and FREE. Such a ‘‘device status’’ type could be defined via (MEMQ BUSY FREE). The new type will be a subtype of the narrowest type that all of the alternatives satisfy (e.g., the "device status" type would be a subtype of LITATOM). The membership test uses EQ if this supertype is a LITATOM; EQUAL otherwise. Thus, lists, floating point numbers, etc., can be included in the set of alternatives.

(ONEOF *TYPE 1. . .TYPE N*) [Type Expression]

Specifies a type that is the union of two or more other types. For example, the notion of a possibly degenerate list is something that is either LISTP or NIL. Such a type can be (and the built-in type LST in fact is) defined simply as (ONEOF NIL LISTP). A union data type becomes a supertype of all of the alternative types specified in the ONEOF expression, and a subtype of their lowest common supertype. The type properties of a union type are taken from its alternative types if they all agree, otherwise from the supertype.

(ALLOF *TYPE 1. . .TYPE N*) [Type Expression]

Specifies a type that is the intersection of two or more other types. For example, a variable may be required to satisfy both FIXP and also some type that is defined as (NUMBERP (SATISFIES *PREDICATE*)). The latter type will admit numbers that are not FIXP, i.e., floating point numbers; the former does not include *PREDICATE*. Both restrictions can be obtained by using the type (ALLOF (NUMBERP (SATISFIES *PREDICATE*)) FIXP).[5]

(OF *AGGREGATE* OF *ELEMENT*) [Type Expression]

Specifies *DECLaggregate,* a type that is an aggregate of values of some other type (e.g., list of numbers, array of strings, etc.). *AGGREGATE* must be a type that provides an EVERYFN property. The EVERYFN is used to apply an arbitrary function to each of the elements of a datum of the aggregate type, and check whether the result is non-NIL for each element. *ELEMENT* may be any type expression. For example, the type ‘‘list of either strings or atoms’’ can be defined as (LISTP OF (ONEOF STRINGP ATOM)). The type test for the new type will consist of applying the type test for *ELEMENT* to each element of the aggregate type using the EVERYFN property. The new type will be a subtype of its aggregate type.[6]

(SATISFIES *TYPE* (SATISFIES *FORM 1. . .FORM N*)) [Type Expression]

Specifies a type whose values are a subset of the values of an existing type. The type test for the new type will first check that the base type is satisfied, i.e., that the object is a member of TYPE, and then evaluate *FORM 1. . .FORM N*. If each form returns a non-NIL value, the type is satisfied.

The value that is being tested may be referred to in *FORM 1. . . FORM N* by either (a) the variable name if the type expression appears in a binding context such as DLAMBDA or DPROG, (b) the distinguished atom ELT for a SATISFIES clause on the elements of an aggregate type, or (c) the distinguished atom VALUE, when the type expression is used in a context where no name is available (e.g., a RETURNS declaration). For example, one might declare the program variable A to be a negative integer via (FIXP (SATISFIES (MINUSP A))) or declare the value of a DLAMBDA to be of type ((ONEOF FIXP FLOATP) (SATISFIES (GREATERP VALUE 25))).

Note that more than one SATISFIES clause may appear in a single type expression attached to different alternatives in a ONEOF type expression, or attached to both the elements and the overall structure of an aggregate. For example,

```
[LISTP OF [FIXP (SATISFIES (ILEQ ELT (CAR VALUE]

           (SATISFIES (ILESSP (LENGTH VALUE) 7]
```

specifies a list of less than seven integers each of which is no greater than the first element of the list.

(SHARED *TYPE*) [Type Expression]

Specifies DECLshared, a subtype of *TYPE,* with default binding behavior, i.e., the binding function (see BINDFN), if any, will be suppressed.[7] For example, if the type FLOATP were redefined so that DLAMBDA and DPROG bindings of variables that were declared to be FLOATP copied their initial values (e.g., to allow SETNs to be free of side effects), then variables declared (SHARED FLOATP) would be initialized in the normal fashion, without copying their initial values.

NAMED TYPES

Although type expressions can be used in any declaration context, it is often desirable to save the definition of a new type if it is to be used frequently, or if a more complex specification of its behavior is to be given than is convenient in an expression. The ability to define a named type is provided by the function DECLTYPE.

(DECLTYPE *TYPENAME TYPE PROP1 VAL1*
PROPN VALN) [Function]

NLambda, nospread function. *TYPENAME* is a literal atom, *TYPE* is either the name of an existing type or a type expression, and *PROP 1, VAL 1...PROP N,VAL N* is a specification (in property list format) of other attributes of the type. DECLTYPE derives a type from *TYPE*, associates it with *TYPENAME*, and then defines any properties specified with the values given.

The following properties are interpreted by the Decl package.[8] Each of these properties can have as its value either a function name or a LAMBDA expression.

TESTFN [Property]

will be used by the Decl package to test whether a given value satisfies this type. The type is considered satisfied if *FN* applied to the item is non-NIL. For example, one might define the type INTEGER with TESTFN FIXP.[9]

EVERYFN [Property]

EVERYFN specifies a mapping function that can apply a functional argument to each ‘‘element’’ of an instance of this type, and which will return NIL unless the result of every such application was non-NIL. *FN* must be a function of two arguments: the aggregate and the function to be applied. For example, the EVERYFN for the built-in type LISTP is EVERY. The Decl package uses the EVERYFN property of the aggregate type to construct a type test for aggregate type expressions. In fact, it is the presence of an EVERYFN property that allows a type to be used as an aggregate type.[10]

BINDFN [Property]

BINDFN is used to compute from the initial value supplied for a DLAMBDA or DPROG variable of this type, the value to which the variable will actually be initialized. *FN* must be a function of one argument that will be applied to the initial value, and which should produce another value which is to be used to make the binding.[11] For example, a BINDFN could be used to bind variables of some type so that new bindings are copies of the initial value. Thus, if FLOATP were given the BINDFN FPLUS, any variable declared FLOATP would be initialized with a new floating box, rather than sharing with that of the original initial value.[12]

SETFN [Property]

is used for performing a SETQ or SETQQ of variables of this type. *FN* is a function of two arguments, the name of the variable and its new value. A SETFN is typically used to avoid the allocation of storage for intermediate results. Note that the SETFN is not the mechanism for the enforcement of type compatibility, which is checked *after* the assignment has taken place. Also note that not all functions that can change values are affected: in particular, SET and SETN are not.

MANIPULATING NAMED TYPES

DECLTYPES is a file package type. Thus all of the operations relating to file package types, e.g., GETDEF, PUTDEF, EDITDEF, DELDEF, SHOWDEF, etc., can be performed on decltypes.[13]

The file package command, DECLTYPES , is provided to dump named decltypes symbolically. They will be written as a series of DECLTYPE forms that will specify only those fields that differ from the corresponding field of their supertype(s). If the type depends on any unnamed types, those types will be dumped (as a compound type expression), continuing up the supertype chain until a named type is found. Care should be exercised to ensure that enough of the named type context is dumped to allow the type definition to remain meaningful.

The functions GETDECLTYPEPROP and SETDECLTYPEPROP, defined analogously to the property list functions for atoms, allow the manipulation of the properties of named types. Setting a property to NIL with SETDECLTYPEPROP removes it from the type.

RELATIONS BETWEEN TYPES

The notion of equivalence of two types is not well defined. However, type equivalence is rarely of interest. What is of interest is type *inclusion*, i.e., whether one type is a supertype or subtype of another. The predicate COVERS can be used to determine whether the values of one type include those of another.

(COVERS *HI LO*) [Function]

COVERS is T if *HI* can be found on some (possibly empty) supertype chain of *LO*; else NIL. Thus, (COVERS ’FIXP (DECLOF 4))= T, even though the DECLTYPE of four is SMALLP, not FIXP. The extremal cases are the obvious identities:

```
(COVERS 'ANY ANYTYPE) = (COVERS ANYTYPE 'NONE) = (COVERS X ) for any type  X
= T.
```

COVERS allows declaration-based transformations of a form that depend on elements of the form being of a certain type to express their applicability conditions in terms of the weakest type to which they apply, without explicit concern for other types that may be subtypes of it. For example, if a particular transformation is to be applied whenever an element is of type NUMBERP, the program that applies that transformation does not have to check whether the element is of type SMALLP, LARGEP, FIXP, FLOATP, etc., but can simply ask whether NUMBERP COVERS the type of that element.

The elementary relations among the types, out of which arbitrary traversals of the type space can be constructed, are made available via:

(SUBTYPE *TYPE)* [Function]

Returns the list of types that are *immediate* subtypes of *TYPE*.

(SUPERTYPES *TYPE*) [Function]

Returns the list of types that are *immediate* supertypes of *TYPE*.

THE DECLARATION DATA BASE

One of the primary uses of type declarations is to provide information that other systems can use to interpret or optimize code. For example, one might choose to write all arithmetic operations in terms of general functions like PLUS and TIMES and then use variable declarations to substitute more efficient, special-purpose code at compile time based on the types of the operands. To this end, a data base of declarations is made available by the Decl package to support these operations.

(DECLOF *FORM*) [Function]

Returns the type of *FORM* in the current declaration context. If *FORM* is an atom, DECLOF will look up that atom directly in its data base of current declarations. Otherwise, DECLOF will look on the property list of (CAR *FORM*) for a DECLOF property, as described below. If there is no DECLOF property, DECLOF will check if (CAR *FORM*) is one of a large set of functions of known result type (e.g., the arithmetic functions). Failing that, if (CAR *FORM*) has a MACRO property, DECLOF will apply itself to the result of expanding (with EXPANDMACRO), the macro definition. Finally, if *FORM* is a Lisp program element that DECLOF ‘‘understands’’ (e.g., a COND, PROG, SELECTQ, etc.), DECLOF applies itself recursively to the part(s) of the contained form which will be returned as value.[14]

DECLOF [Property]

Allows the specification of the type of the values returned by a particular function. The value of the DECLOF property can be either a type, i.e., a type name or a type expression, or a list of the form (FUNCTION *FN*), where *FN* is a function object. *FN* will be applied (by DECLOF) to the form whose CAR has this DECLOF property on its property list. The value of this function application will then be considered to be the type of the form.

As an example of how declarations can be used to automatically generate more efficient code, consider an arithmetic package. Declarations of numeric variables could be used to guide code generation to avoid the inefficiencies of Interlisp's handling of arithmetic values. Not only could the generic arithmetic functions be automatically specialized, as suggested above, but by redefining the BINDFN and the SETFN properties for the types FLOATP and LARGEP to reuse storage in the appropriate contexts (i.e., when the new value can be determined to be of the appropriate type), tremendous economies could be realized by not allocating storage to intermediate results that must later be reclaimed by the garbage collector. The Decl package has been used as the basis for several such code optimizing systems.

DECLARATIONS AND MASTERSCOPE

The Decl package notifies MASTERSCOPE about type declarations and defines a new MASTERSCOPE relation, TYPE, which depends on declarations. Thus, the user can ask questions such as "WHO USES MUMBLE AS A TYPE?," " DOES FOO USE FIXP AS A TYPE?," and so on.

END NOTES

1. USEDIN is mainly for documentation purposes, since there is no way for such a restriction to be enforced.

2. Like USEDIN declarations, FREE and BOUNDIN declarations cannot be checked, and are for documentation purposes only.

3. With this exception, assignments to variables from within the break are not considered to be in the scope of the declarations that were in effect when the break took place and so are not checked.

4. LST is defined as either LISTP or NIL. i.e., a list or NIL. The name LST is used because the name LIST is treated specially by CLISP. A LIST is defined as either NIL or a list of elements each of which is of type LISTP.

5. When a value is tested, the component type tests are applied from left to right.

6. The built-in aggregate types are ARRAP, LISTP, LST, and STRINGP (and their subtypes).

7. As no predefined type has a binding function, this is of no concern until the user defines or redefines a type to have a binding function.

8. Actually, any property can be attached to a type, and will be available for use by user functions via the function GETDECLTYPEPROP.

9. Typically, the TESTFN for a type is derived from its type expression, rather than specified explicitly. The ability to specify the TESTFN is provided for those cases where a predicate is available that is much more efficient than that which would be derived from the type expression. For example, the type SMALLP is defined to have the function SMALLP as its TESTFN, rather than (LAMBDA(DATUM) (AND(NUMBERP DATUM)(FIXP DATUM) (SMALLP DATUM))) as would be derived from the subtype structure.

10. Note that a type's EVERYFN is not used in type tests for that type, but only in type tests for types defined by OF expressions that used this type as the aggregate type. For example, EVERY is not

used in defining whether some value satisfies the type LISTP. The Decl package never applies the EVERYFN of a type to a value without first verifying that the value satisfies that type.

11. For a PPROG binding, FN will be applied to no arguments if the initial value is lexically NIL.

12. The BINDFN, if any, associated with a type may be suppressed in a declaration context by creating a subtype with the type-expressing operator SHARED.

13. Deleting a named type could possibly invalidate other type definitions that have the named type as a subtype or supertype. Consequently, the deleted type is simply unnamed and left in the type space as long as it is needed.

14. ‘‘The current declaration context’’ is defined by the environment at the time that DECLOF is called. Code-reading systems, such as the compiler and the interpreter, keep track of the lexical scope within which they are currently operating, in particular, which declarations are in effect. Note that (currently) DECLOF does *not* have access to any global data base of declarations. For example, DECLOF does not have information available about the types of arguments of, or the value returned by, a particular function, unless it is currently ‘‘inside’’ that function. However, the DECLOF property can be used to inform DECLOF of the type of the value returned by a particular function.

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 explicity 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.

```
DEMO -- utilities for running demos / tutorials in Medley

includes
   OPEN-URL (URL)
                 (rename of ShellBrowse)
   MEDLEY-CONTRIB(REPO)
                  shows GitHub contributors to given repo
                  uses ShellBrowse
    BKSYSOBJ(string)


DEMO-*.TEDIT
    contains scripts / TEDIT file talks
    add your own



BKSYSOBJ is the start of a facility

(TEDIT.INSERT.OBJ (BKSYSOBJ "(CONS NIL")     (TEXTSTREAM(WHICHW)]



You should see (CONS NIL) in the TEDIT stream, clicking should shove (CONS NIL into an Interlisp
exec, waiting for ) or ].  (probably the image objectg should be shaded, may also have to set the
RDTBL flag on BKSYSBUF for strings, but this is a start).
```

Medley Interlisp

March 2023

Interlisp.org

- DEMO-OVERVIEW (this file)
- DEMO-FEATURES (maybe other DEMO-FEATURE-XXX)
- DEMO-PROJECT (what have we been doing / demos)

What is Medley Interlisp?

- The software for the Xerox Lisp machines, developed from 1960's (BBN), thru 1970's and 80's (Xerox), and 90's (Envos, Venue)

- ACM Software System Award 1992

  The features of ***structure editing****,* ***source code management****,* ***code analysis*** *and* ***cross-referencing*** *combined to support* ***rapid incremental development****. The 1992 ACM Software System Award was awarded to the Interlisp system for pioneering work in programming environments.*

- Then: expensive, slow, unwieldy, and ... unavailable

- Now: on modern hardware of all sizes, 1000 times faster

Medley Interlisp is an IDE

In Interlisp, you could refer to a function you hadn't written yet, run your code until it broke, and from the break have it pop open an editor with the signature of your function. You could also inspect the values of the arguments passed. You could then write the function and continue the computation.

Similarly if your code broke with an error, you could edit the function on the stack to correct the error, and continue the computation. ...

When demonstrating this capability in Interlisp.the response was along the lines of "why would I ever want that?" ... If you've never used an environment focused on programmer productivity you have no idea how to even think along those lines.   -- Simon Brooke

Medley Interlisp

Features Demos

A Residential Programming Environment supports Incremental Development

A residential programming environment is one in which you define, edit, debug in the live environment; write out to files to save for future session. The system tracks what changed. Multiple features work together, to act as a "programmer's assistant":

DWIM - Do What I Mean, Spelling Correction: DWIM is more than just spelling correction; it's more an attitude -- not to signal errors if what the user meant is obvious and correctable.

```
(DEFINEQ (FOO1 (X) (PLUSS X X]      <-- PLUS misspelled.
                                    < Note []
superparenthesis.
PP FOO1
(FOO1 6]
Y
PP FOO1
```

History and UNDO: Interlisp was the first known system. DWIM changes are undoable.

```
??
UNDO <dwim event>
PP* FOO1
```

- The file manager tracks what's been defined or changed. The file "coms" are like a manifest -determines what goes where.

```
FILES?)
```

```
y FILEB
(CLEANUP)
SEE FILEB
```

- *Helpsys and DInfo (also a first) interactive documentation*

```
MAN INFILE
MAN CL:WITH-OPEN-FILE
MAN CLHS.OPENER
<right click DInfo>
```

- *Masterscope (interactive cross reference) has an extensible english-like language for querying*

```
LOAD(HELPSYS PROP)
. ANALYZE ON HELPSYS
. SHOW WHERE ANY CALLS CLHS.OPENER
USE EDIT FOR SHOW
. SHOW PATHS FROM HELPSYS
```

Common Lisp and Interlisp Integration

an Inter-medley of Uncommon Lisps

- freely intermix CL and IL function, macro, variable definitions

- All datatypes are common: lists, NIL, symbols, arrays, strings, structures, numbers

- many functions and special forms are the same: CAR, COND, ED

- Some are slightly different: CL:EVAL vs. IL:EVAL, CL:LAMBDA vs. IL:LAMBDA, ...

- Extend CL to include IL and IL to include CL

- Common Lisp definitions and declarations managed by Interlisp environment

Demo:

```
. SHOW WHERE ANY ON HELPSYS CALLS CL:WHEN
USE EDIT FOR SHOW
```

Structure editors

`EDITMODE(TELETYPE)` original structure editor: powerful, programmable, short commands (BO 3) P (SW 1 2). Useful for editing huge list structures or Lisp functions.

`EDITMODE(DEDIT)` (Load it to try). Like the TTY editor with menu and showing the results in a window.

`EDITMODE(SEDIT:SEDIT)` Default editor. SEdit maintains illusion that you are editing structure. Parentheses are balanced at all times, but you can just type and backspace. Keyboard controls and attached menu.

Common to all: user never counts parentheses, modifies whitespace, or needs to fiddle with line breaks or indentation

The Virtual Machine and OS

- D-machines had microcode to interpret a bytecoded insruction set. Subsequently, the microcode was reimplemented in C (named"maiko".

- "Sysouts" (memory images) can be moved from machine to machine; only maiko needs to have been compiled for each OS and chip architecture.

- **Medley is small** (relatively speaking). Bytecodes are compact. Medley online uses 16MB / user (64MB max). Installed, 256mb max. Time to make new image 15 seconds; restart a saved image "in the blink of an eye".

- Interlisp-D was the **whole operating system**: scheduler, window manager, network, drivers. Now can rely on host OS for device drivers

GIT: A Repository for Medley Definitions

Conventionally, GIT tracks and compares files

- Change detection and presentation based on line-editing semantics: Mismatching character sequence => significant difference

Medley's source files: external archives for structured definitions with metadata

- Saved and loaded, maybe printed, but never edited

- A given definition can be represented in different (but semantically equivalent) line and character sequences

Medley interface to GIT: Definition-based change tracking

P(ull)**R**(equest)**C**(ompare)   command   (demo)

- Retrieve changed files from GIT, find / parse PR vs master alternative definitions, compare as Lisp-structure differences

Idea: load and manage definitions @ commit level granularity

- If function FOO is included in commits C and D, definitions FOO;C and FOO;D are co-resident.

LispUsers, Library, Internal

(Favorite LispUsers)

Some other development tools: SPY, File Browser

Revive Applications using Interlisp

Notecards - early HyperText

Rooms - Screen / Desktop management

LOOPS - Lisp Object Oriented Programming System (not CLOS)

Truckin’ - LOOPS game for teaching "Knowledge Representation"

LFG - Lexical Functional Grammar

.... and others

Medley Interlisp

2023 Project 18 March 2023

Interlisp.org

What have we been doing?

+ Adapting the system to current environments and standards

    hardware: keyboard, mouse, display, CPU (64bit, little endian)

    software: standard C, Unicode, Posix, UI Guidelines

+ Lower barriers to entry

+ Demonstrate unique and original features

+ Help with revival of other applications built using system

+ Improve maintainability for future users

+ Gather and update documentation

Who might use Medley Interlisp?

- Retrocomputing enthusiasts

- Researchers, IDE tool creators, looking into ideas

- Software and AI historians (present and future)

- Software archivists (SPN, technical infrastructure)

- Developers of new applications

Who is involved?

+ Some of original developers from Xerox PARC (in 70’s)

+ Developers of s using Interlisp (recursive revivers)

- Open Source contributors

- Students of computer science and history

+ Friends, enthusiasts, and ... you?

```
(OPEN-URL "https://github.com/orgs/interlisp/people")
(MEDLEY-CONTRIB "medley")
(MEDLEY-CONTRIB "maiko")
(MEDLEY-CONRIB "online")
(MEDLEY-CONTRIB "Interlisp.github.io")
```

More on Modernizing Medley

Goals

- Adapt Medley UI to current hardware

- Make Medley work more like current applications for comfort

Mouse: Wheel Scroll, Window move on title, Window resize on corner. Warning: Exec windows. Consolation: you can do it all with menus. Reduction: some menus are invoked by "middle" mouse button which on some mice is hard to press.

Keyboard: Desired state: compatible keystrokes. Subtasks: decide what keystrokes do what; slash through umpteen layers of keyboard re-iterpretation.

Display: Color, high-resolution displays, modern fonts.

Reduce Barriers to Entry

- Running Online

```
OPEN-URL("https://www.youtube.com/watch?v=mI3Ga5LyIlI")
OPEN-URL("https://online.interlisp.org")
```

- Installation Instructions

```
OPEN-URL("https://interlisp.org/running/")
```

More notes on Running on Modern Systems

- about K&R C and C standards; type declarations, big-endian vs. little endian, 32 bit vs 64 eliminating errors, file-sytem changes, process handles, bugs left over from 24- to 28-bit address space

- docker, installers, CO / CI(?), two levels of virtualization

- online.interlisp.org: connect in seconds.

- Mysteries in the code we inherited, "software archeology"

- Robust "loadup process", all day vs seconds

- bytecode virtual machine aids portability.

```
fb {li}<maiko>bin/makefile-*.*-x
```

Linux, Mac, Windows (WSL1, WSL2, Cygwin)

Docker, FreeBSD, OpenBSD, SunOS5
x86_64, i386, arm7l, arm64, and older

Getting Involved

- *Try things out, report new problems, or new reproducible cases*

- *Clean up old issues*

```
(OPEN-URL "https://github.com/interlisp/medley/issues")
```

```
(FILESLOAD PICK)
```

```
PICK ISSUE
```

- *Network emulation*

-*Donate? See our GitHub sponsor page*

DIALPHONE

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

INTRODUCTION

DIALPHONE is a simple computer-controlled telephone dialer. It requires a modem connected to the RS-232 port. It should work with a wide variety of modems since it makes no assumption about any return codes from the modem. It lets you dial your phone by typing (or selecting) a number fromthe computer's keyboard. A history list of recently dialed numbers is maintained. It knows how to deal with differences between dialing local extensions, outside calls, and long distance. It also translates phone numbers containing letters back into numbers.

OPERATION

Load DIALPHONE.LCOM from your local LispUsers directory and call (DIALPHONE). The program will ask you for your own phone number. This is used within Xerox for billing long distance calls. If you just enter a CR, the program will not append any number at the "quick-quick-quick-slow" tone when dialing long distance. Next, it will try to attach a menu with a telephone icon to an AddressBook window (if you have the LispUsers package AddressBook loaded), since it makes a useful complement to AddressBook. If you do not have AddressBook loaded, it will prompt you to click in some window where you would like the Dial icon attached. If you click in an area of background, a stand-alone menu will be created. Before dialing a number, check the various paramters described below.

To dial a number, click on the phone icon with the left mouse button. You will see the prompt "Number please:" in the prompt window. You may now either type in a number or shift-select it out of another window. To abort the operation, type a CR without a number. If you select the icon with the middle button, you will get a menu of the last 10 numbers dialed. Listen to the number being dialed over the modem's speaker, then pick up the receiver. The program will make the modem hang up automatically shortly after the number is dialed.

The program converts any letters in the number you give it into numbers. The program is designed for use on a PBX-style system (although it is easy to modify if desired). Numbers less than six digits long are assumed to be internal extensions and are dialed exactly as given. Numbers or 7 or 8 digits are assumed to be local outside calls; the program prefixes a "9" to them. Numbers longer than 8 digits are assumed to be long distance; the program appends your extension to them for billing. Xerox Intelent numbers should be typed in their usual form, starting with "8*". Area codes and exchanges should be separated with dashes, e.g. "716-555-1212".

You can also call the program directly, e.g. (DIALPHONE "555-1212"). It will return the number dialed. If you try to dial a number while the modem port is already in use, the program will print an error message in the Prompt Window and return without dialing.

DIALHISTSIZE [Variable]

This controls how many numbers will appear in the menu you get when you select the phone icon with the middle button. Default = 10.

DIALPREFIX [Variable]

The modem command your modem needs to initiate dialing. Default = "ATDT". Change this to "ATDP" if you do not have touch-tone service.

DIALSUFFIX [Variable]

The modem command your modem needs to terminate a command. Default = <CR> (ASCII 13).

LASTNUMBERDIALED [Variable]

The last number you dialed. initially NIL.

PHONEBILLNUMBER [Variable]

The number the program should use to append to long-distance numbers. Initially "" (null string).

DICTTOOL

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

Uses DICTCLIENT, ANALYZER

INTERNAL

INTRODUCTION

DICTTOOL is the user’s interface to the Dictionary Server. The Dictionary Server is a prototype of a shared network resource for providing a suite of dictionary-based capabilities to programs running on client workstations. It has on it the American Heritage dictionary, the Word Finder synonym package by Microlytics, a Proofreader, and the WordNerd, a package for searching for words based on their meaning.

(Note: The American Heritage dictionary has been licensed to us by Houghton-Miflin for research purposes only, and so we have not made the Dictionary Server generally available. The Dictionary Server should only be used by people within PARC.)

HOW TO USE DICTTOOL

When you load the DICTTOOL, it automatically adds a new menu item named "Dictionary" to the TEdit menu and the Background menu. The "Dictionary" menu item has three sub-items: "Get Definition", "Get Synonyms", and "Search For Word". Here is how each one works:

Get Definition

If you make a selection in a TEdit document, and then invoke the "Get Definition" command in that document, then DictTool will ask for a confirmation and then fetch the definition for that word from the Dictionary Server, printing it in a separate window. If there is more than one entry in the American Heritage Dictionary for that word, then it will print the definitions one after another. The Dictionary Server knows how to find the root forms of words, and so "breathing" "breathes" and "breathe" will all give you the same entry.

If there is no selection in the TEdit document, or if you deny the confirmation, or if instead of using the TEdit menu you use the Background menu, then DictTool will first prompt you for a word to look up and then fetch its definition. (Since it is very hard to make a null selection with TEdit, DictTool treats a one character selection as meaning "no selection". If you really want to look up a single letter in the dictionary, you can type it in when prompted for a word.)

If you want to look up several definitions at once, separate the entries with semi-colons followed by spaces. (i.e. "camera; photography; motion picture"). Semi-colons are used as delimiters because some of the entries in the American Heritage Dictionary have spaces and commas in them (as in "motion picture"). It also makes it easier to look up words in the output of the WordNerd (see "Search For Word" below).

Get Synonyms

The interface for getting synonyms is exactly the same as the interface for getting definitions. If you make a selection, then DictTool will first confirm the selection and then print out the synonyms in the same window that the definitions are printed in. If you don't make a selection, then DictTool will first prompt you for a word. The format of the information printed out is a series of synonym classes separated by carriage returns. Each synonym class begins with the part of speech that its elements belong to. The elements themselves are separated by commas.

Search For Word

The interface to the WordNerd is a little different from the other interfaces. Instead of typing just one word in, you want to type a list of keywords separated by spaces. For instance, if you were looking for the word for a mechanical model of the solar system, you might type:

Type keywords to search on: mechanical model solar system

The WordNerd then searches for words that have at least two of these keywords in their definitions. The results would be sorted according to the number of keywords found, with the words having the most keywords printed first:

mechanical model solar system: orrery

mechanical solar system: mechanism

model solar system: planetarium

mechanical system: automation; bar1; component; degree of freedom; energy level; hookup; ignition; instrument; key1; linkwork; load; machine; neutral; perpetual motion; quantize; resonance; schematic; servomechanism; shafting; stress; suspension; unit

solar system: Copernican; cosmic; Earth; Ganymede3; Jupiter2; Mars2; mercury; Milky Way; nebular hypothesis; Neptune3; Pallas; planet; planetesimal hypothesis; Pluto2; Saturn2; solar battery; space; sun; Uranus2; Venus2; Vesta2

(The numbers after some of the words mean that this is the nth entry of this word in the American Heritage Dictionary.) If there is a word in the list that you want to see the definition for, you can merely select it and get its definition with "Get Definition". In this case you would probably want to know what "orrery" means:

> **or|re|ry** *n., pl.* **-ries.** A mechanical model of the solar system. [After Charles Boyle (1676–1731), fourth Earl of *Orrery,* for whom one was made.]

There is also a mechanism for indicating that two words are synonyms of one another, and hence should not be counted as separate keywords for the purpose of deciding whether a word has the minimum two keywords. All you need to do is put parentheses around the words in question. For instance if you were looking for the word for the little plastic thing on the end of a shoe lace, you might try:

> *Type keywords to search on*: (shoe lace shoelace) (end tip)

And get in return:

> **shoe+ end+:** aglet; fall; heel1; lift; point; quarter; spike1; toe

(A plus at the end of a word indicates a synonym class.)

If you only give the DictTool one word, then it will print out all of the words in the dictionary that have that word in its definition.

Max Words

There are two sub-items in the "Search For Word" sub-menu: "Max Words" and "Min Keywords". The first sub-item, "Max Words", allows the user to specify the maximum number of words that should be returned on each search. DictTool is set up to only return 100 words at a time. If WordNerd finds more than a hundred words, then it truncates the list and indicates how many words it eliminated. If you want to see the words that were eliminated, just make the same request with the same keywords in the same order and the WordNerd will return the next 100 words. (If there is no selection in the document, then DictTool will prompt you with the last set of keywords so that this is easier.) However, if 100 words is too many or too few, you can change it with this menu item or by setting the global variable DictTool.MaxWords.

Min Keywords

DictTool is set up to only return a word if it has at least two of the user's keywords in its definition. If the user wants, he can raise or lower the minimum as he sees fit. The minimum only comes into play whenever the user gives more keywords than the minimum, otherwise the WordNerd looks for words that have at least one of the keywords in their definition. A minimum of 1 means that only one word has to match. A negative minimum means that the WordNerd will set the minimum relative to the number of keywords given. For instance, a minimum of -1 says that all but one of the keywords have to match for the word to be returned. A minimum of zero means that *all* of the words have to match.

Search For Phrase

The Search For Phrase command returns all of the entries in the American Heritage Dictionary that have a particular phrase in them. It does this with the help of the Search For Word command, which is why it is a sub-command of that command. Whenever you search for a phrase, the dictionary server first uses the Search For Word command to get the list of words in the dictionary that have all of the words of the phrase in it. It then looks up the definition of each of these words, and returns the words that have the phrase in their definition. This can be a very time-consuming operation, so you should use this command sparingly. But if you are concerned about locality and word order, then this command can save you a lot of time.

PROOFREADING

The Dictionary Server also provides proofreading facilities similar to the PROOFREADER package. The interface is exactly the same: there is a "Proofread" menu item on the TEdit menu which produces a special fixed menu for proofreading. The only difference is that all of the proofreading is done remotely on the server. You should only use the Dictionary Server for proofreading small documents; if you are going to do a lot of proofreading, it is better to use the PROOFREADER. (For more documentation on how to proofread, see PROOFREADER.)

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 slected. 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.

DINFO

By: Doug Cutting (Cutting.PA@Xerox.COM)

Uses: GRAPHER, TEDIT

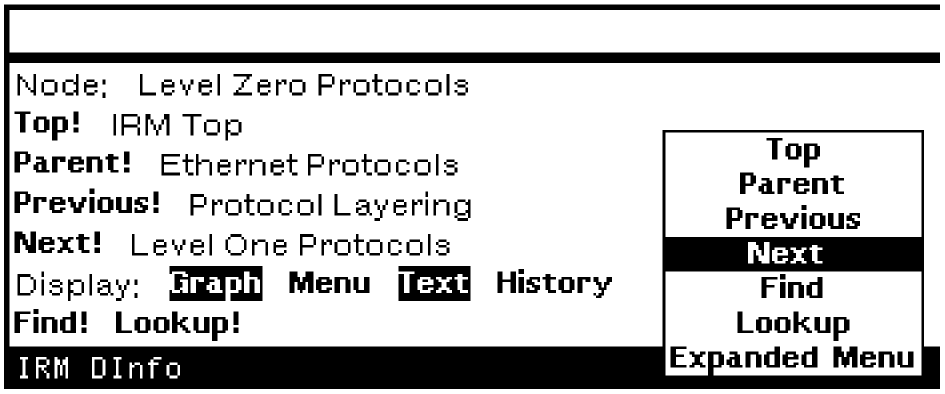
This document last edited on October 7, 1987.

INTRODUCTION

DInfo is a system for browsing graph structured documentation. Graphs for the *Interlisp-D Reference Manual* and the *Xerox Quintus Prolog Manual* are available, as are tools for creating and editing new graphs.

USER INTERFACE

Selecting DInfo from the background menu will pop up a menu listing the available graphs (see DINFO.GRAPHS below). Selecting one of these items will bring up a browser on that graph. Most interaction with DInfo is done through menus at the top of each graph browser which look like this:


Node: Level Zero Protocols
Top! IRM Top
Parent! Ethernet Protocols
Previous! Protocol Layering
Next! Level One Protocols
Display: Graph Menu Text History
Find! Lookup!
IRM DInfo
Top
Parent
Previous
Next
Find
Lookup
Expanded Menu

The Expanded Menu may be closed independently of the graph browser, and re-opened by selecting **Expanded Menu** from the title bar menu. Except for this command, the commands in the title bar menu are identical to their counterparts in the Expanded Menu, so only the latter commands are documented.

Next to **Node:**, **Top!**, **Parent!**, **Previous!**, and **Next!** are printed the names of the node currently being visited, the name of the top node in the graph, the name of the node previous to the current node, and the name of the node next to the previous node, respectively. **Top!**, **Parent!**, **Previous!**, and **Next!** are also commands as follows:

Top! [DInfo Menu Command]

Visits the top node in the graph.

Parent! [DInfo Menu Command]

Visits the parent of the current node.

Previous! [DInfo Menu Command]

Visits the node previous to the current node.

Next! [DInfo Menu Command]

Visits the node following the current node.

Previous! and **Next!** thus provide sequential access to the graph.

The Display: toggles control what will be displayed when a node is visited:

Graph [DInfo Menu Toggle]

toggles display of a Grapher display of the graph local to the current node. Selecting a node in this display will visit the corresponding node in the graph.

Menu [DInfo Menu Toggle]

Toggles display of a menu of subnodes of the current node. If the current node has no subnodes no menu will be displayed. Selecting an item in the subnode Menu will visit that node in the graph.

Text [DInfo Menu Toggle]

Toggles display of the text of the current node. Turning this off will speed up the visiting of nodes considerably, useful when searching for a particular node.

History [DInfo Menu Toggle]

Toggles display of a menu containing the history of nodes visited. Selecting an item from this menu will revisit that node.

Lookup! [DInfo Menu Command]

If selected with the left mouse button prompts for a term and then calls the LOOKUPFN of the graph with it. Using the middle button will re-call the LOOKUPFN with whatever was used previously. This is intended for the lookup of terms in a graph-dependent index. In the case of the *Interlisp-D Reference Manual* DInfo Graph, Lookup! will lookup a term in the index of the IRM, and then visit the node containing the reference to this term.

(DINFO *GRAPH.OR.FILE WINDOW.OR.REGION* — —) [Function]

Starts a DInfo browser on GRAPH.OR.FILE in WINDOW.OR.REGION. If GRAPH.OR.FILE is NIL, an empty graph will be created. If WINDOW.OR.REGION is NIL, the user will be prompted for a window or region.

GLOBAL VARIABLES

DINFOMODES [Variable]

Determines which of the toggles will be selected when DInfo is initially started; it should be a list with recognized members being GRAPH, MENU, TEXT, and HISTORY. Default is (GRAPH TEXT).

DINFO.HISTORY.LENGTH [Variable]

Determines the maximum length of DInfo's history. Default is 20.

DINFO.GRAPHS [Variable]

Determines the contents of the menu raised by selecting DInfo from the background menu. Should be a menu-items style list where when the CADR of an item is evaluated it returns either a Dinfo Graph (a DINFOGRAPH record) or the name of a file containing a DInfo Graph (i.e, something suitable for passing to DInfo as the GRAPH.OR.FILE arguement).

INTERNALS

The following information is included for the programmer interested in adding alternate graphs to DInfo.

DINFONODE [Record]

Contains the following fields:

ID Unique identifier for node in graph, ala GRAPHNODE field NODEID. Note that EQ is used for checking identity of nodes.

LABEL The print name of a node. Analagous to the GRAPHNODE field NODELABEL.

FILE The file containing the documentation for this node. Should not generally include HOST and DIRECTORY fields as DInfo will default these (assuming all documentation files are on one directory, see below).

FROMBYTE Byte number in FILE where the documentation for this node begins.

TOBYTE Byte number in FILE where the documentation for this node ends.

DInfo uses OPENTEXTSTREAM to display its files, and thus any TEdit file can be included. Note that if a file has any formatting (image objects in particular), the byte number of a character in a file is not necessarily the same as the TEdit character number of that character.

PARENT The ID field of the node parent to this node.

CHILDREN A list of the ID's of the subnodes of this node.

NEXTNODE The ID of the next node in the graph.

PREVIOUSNODE The ID of the node previous to this node in the graph.

USERDATA Unused. Note that there is no special access function for this field as, for example, WINDOWPROP is for the USERDATA field of a WINDOW. This field is left open for use by implementors for whatever they see fit.

DINFOGRAPH [Data Type]

Contains the following fields of interest to the implementor:

NAME The name of the graph. Note that when DInfo reads a graph from a file (with DINFO.READ.GRAPH) this field is set to the NAME field of the file name the graph is read from.

NODELST The list of nodes in the graph. Each node should be a DINFONODE record.

TOPNODEID The ID field of the root, or top node of the graph.

WINDOW The main window of the graph browser.

CURRENTNODE Used by DInfo to keep track of where in the graph it is.

DEFAULTHOST Used if no host is specified in the FILE field of a node.

DEFAULTDEVICE Used if no device is specified in the FILE field of a node.

DEFAULTDIR Used if no directory is specified in the FILE field of a node.

Note that DEFAULTHOST, DEFAULTDEVICE and DEFAULTDIR are set when a DINFOGRAPH is read from a file (by DINFO.READ.GRAPH) to the host, device and directory of that file.

TEXTPROPS Will be passed as the PROPS argument to OPENTEXTSTREAM when the file for a node in the graph is displayed. This feature can be to used to fake some formatting .

LOOKUPFN Will be called when the user selects Lookup! from DInfo's Expanded Menu with two arguments: The string to look up, and the current DInfo graph.

MENUFN Called when the middle mouse is depressed in the title bar of a graph's window . If not specified, DINFO.DEFAULT.MENU will be used. Passed one argument of the current DInfo graph. DINFO.EDIT.MENU is a MENUFN that allows editing of DInfo graphs. Selecting >>Empty Graph<< from the menu raised by selecting DInfo from the Background Menu will start DInfo on a an empty graph with this MENUFN.

FREEMENUITEMS DInfo's Expanded Menu is implemented as a FreeMenu. This property holds a list of FreeMenu item descriptions suitable for passing to FREEMENU (see FreeMenu documentation). This list will be appended onto the bottom row of buttons (**Find!** and **Lookup!)** whenever a FreeMenu is created for this graph.

USERDATA Accessed by the macro DINFOGRAPHPROP. See below.

(DINFOGRAPHPROP *GRAPH PROP VALUE*) [Macro]

If VALUE is not specified, will return the PROP of GRAPH. PROP can be either a real DINFOGRAPH field or something in the USERDATA field. If VALUE is specified it will be put in GRAPH at PROP. Note that in this case it will return the *new value* stored in, not the previous value, as many other Interlisp-D access functions do.

(DINFO.UPDATE *GRAPH NODE SEL*) [Function]

Will visit NODE in GRAPH. *NODE,* if specified*,* should be a DINFONODE record which is in the NODELST of GRAPH. *SEL* is used by DInfo's *Interlisp-D Reference Manual* lookup facility, and should be useful in implementing other lookup facilities. *SEL* determines what in the TEXT of this node will be selected. *SEL* should be a string or a list of the format (*NAME X*) where *NAME* is the name of the selection, *X* is the character number in the text of *NODE* to be scrolled to. If SEL is a string, the string will be searched for in the text of NODE and selected. This is useful for the lookup of terms.

(DINFOGRAPH *WINDOW*) [Function]

Return the DINFOGRAPH associated with window.   Note that the pointer from the window to the graph is destroyed when the window is closed to remove circularity. For this reason it is better to keep a handle on the DINFOGRAPH and use (DINFOGRAPHPROP <DInfoGraph> 'WINDOW) when you need the WINDOW.

(DINFO.READ.GRAPH *FILE QUIETFLG*) [Function]

Reads a file written by DINFO.WRITE.GRAPH, and returns the DINFOGRAPH contained therein. If QUIETFLG is non-NIL, nothing will be printed out while reading.

(DINFO.WRITE.GRAPH *GRAPH FILE*) [Function]

Writes GRAPH to FILE such that it can be read by DINFO.READ.GRAPH.

(DINFO.READ.KOTO.GRAPH *GRAPH FILE*] [Function]

Reads a file written by Koto DINFO.WRITE.GRAPH and returns a Lyric DINFOGRAPH.   Thus:

```
(DINFO.WRITE.GRAPH
  (DINFO.READ.KOTO.GRAPH <file1> T)
  <file2>)
```

will convert the Koto format DINFOGRAPH in `<file1>` to a Lyric format DINFOGRAPH in `<file2>`.

DIR-TREE

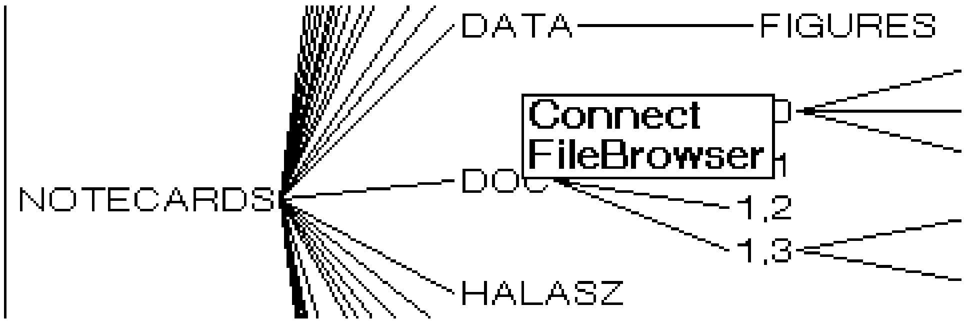
Dir-Tree builds directory trees using the grapher package.

It works on IFS, XNS, and NFS devices from Medley.

Loading Dir-Tree puts a menu option on your background menu.

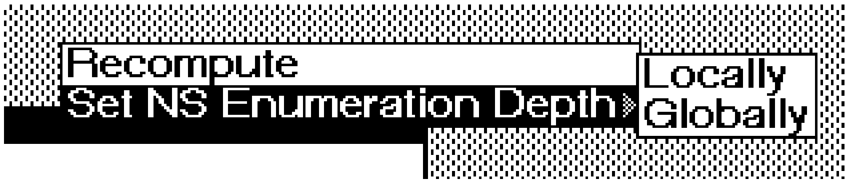
Dir-Tree can take awhile to compute a large directory. Do not give it a /users directory to graph.

From the directory window you can left button on a node and bring up a menu to Connect to the directory or call FileBrowser on the directory.



At the moment shift-select from the graph does not work but you can achieve the same effect by clicking the middle mouse button on a node.

Holding the left mouse button down in the title bar brings up the recompute option. On NS servers it also allows you to set the Enumeration Depth.



I make no pretenses that Dir-Tree is anything like a finished product.
However, it has been satisfactorily working for me in its present state for over a month now (Jan '89).

I have been asked when it will be available. Since I will not have any time to work on it until the NoteCards doc is done, those who are interested are welcome to use it in its present state.

KNOWN PROBLEMS:

When using it from a Sun DO NOT use the {UNIX} device. This hopelessly confuses Dir-Tree.

When specifying the root directory on a Sun, you must get the case right. Dir-Tree will still generate a graph but it will only go down one level, and it will not correctly truncate the root directory name in the graph window.

Keith Mountford.

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 1988 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 withthe following fields: "New File" (prompt for a new file name), "Edit File" (TEdit the Include file, or bring it to the top if 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 ofone 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 *Excruciatingly* 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 trys 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 exand 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.

DORADOCOLOR

Maintained By: Frank Shih (Shih.envos@Xerox.com)

INTERNAL

Uses: COLOR

This document last edited on 8-Nov-88.

INTRODUCTION

This package is the Xerox Lisp software driver for the Dorado (Xerox 1132) color display.

NECESSARY HARDWARE

You need a Xerox 1132 with color card and a third party color display. Please contact your Xerox representative for details concerning acquiring and setting up all the required hardware. Some notes on configuring the Conrac color monitor are given at the end of this document.

Assuming you have all the hardware you need, turn it all on. This means

(1) Your 1132 is running Xerox LISP.

(2) Your 1132 has an 1132 color card installed.

(3) Your color display is plugged in and powered on.

(4) Three cables for red, green, and blue signal connect the 1132 color card to the color display.

Any reconnections should be made while your 1132 is off. Until you issue some software commands, a black color display is normal.

DORADOCOLOR SOFTWARE

The DORADOCOLOR package provides the machine dependent portion of software that is needed to drive your color display assuming you are using an 1132 with 1132 color card. Other than LOADing the DORADOCOLOR package and turning the DORADOCOLOR package on using the function COLORDISPLAY, all additional functionality is provided by and documented with the COLOR package. There are no DORADOCOLOR functions that the user needs to call directly. The user calls functions described in the COLOR documentation. A single exception to these comments is a global variable \DORADOCOLOR.LEFTMARGIN which controls the period of time the display controller should wait before turning on the color guns. \DORADOCOLOR.LEFTMARGIN is normally set to 56; if this value causes odd results with your monitor, try setting \DORADOCOLOR.LEFTMARGIN a little higher or lower and reinitializing the display. (You can reinitialize the display by calling COLORDISPLAY twice in succession).

Once your hardware is on, you can proceed to issue COLOR commands to your hardware. You should have the DORADOCOLOR package already LOADed from your LIBRARY directory. That is, you've already done something like (FILESLOAD DORADOCOLOR). At this point it may be convenient to follow this documentation along with the documentation for COLOR in the Lisp Library Packages Manual. If you now type

```
(COLORDISPLAY 'ON 'DORADOCOLOR)
```

your display will now change from total black to a color test pattern with horizontal and vertical stripes. The sequence of events is that there should be a noticable flicker on your color display, followed by a white wall covering the color display, followed by the painting of this white wall with horizontal and vertical strpes of color woven together. There are now some simple tests you can do to satisfy yourself that your hardware is working. Here is a small list of things to try:

```
        (SETQ CSBM (COLORSCREENBITMAP))

        (BLTSHADE 'WHITE CSBM)

        (BLTSHADE 'RED CSBM)

        (BLTSHADE 'GREEN CSBM)

        (BLTSHADE 'BLUE CSBM)

        (SETQ DS (DSPCREATE CSBM))

        (DRAWLINE 0 0 500 500 10 'REPLACE DS 'YELLOW)

        (DRAWLINE 500 0 0 500 10 'REPLACE DS 'CYAN)
```

Assuming all has gone well to this point, you should now be able to try all the functions described in the COLOR package documentation. The COLORDEMO package is a good source of test programs to try — (IL:LOAD 'COLORDEMO.LCOM) to get this package. Both COLOR and COLORDEMO documentation are in your LispUsers' Manual.

KNOWN BUGS

As of 11/88, there are several known bugs with the color code. Dragging the mouse off the right hand edge of the display appears to hang the Dorado. Also, color fonts do not seem to work (probably because in Medley AC fonts are unrotated using pilotbbt) .

APPENDIX - THE CONRAC COLOR MONITOR

This section describes configuring the Conrac color monitor (model 7211C19) for use with the Dorado**.**

Back Panel - Connections need to be made as follows: each color cable (red, green, blue) should be connected to the color-corresponding IN terminal. The black cable should be connected to the SYNC/HDRIVE IN terminal. The rocker switches next to the connectors should be set to 75 ohms.

Front Panel - The BRIGHT and CONTrast knobs push in to configure to their preset settings, pull out for adjustment. The SCREEN buttons should be toggled out, the CHANNEL button should be toggled out.

Internal Switches - If after connecting the monitor and starting up the color software, the display does not appear to be in sync, you may need to check the internal settings inside the monitor. To do so, you will need to remove the cover and several internal metal panels (which may be screwed to the bottom of the monitor as well). Viewed from the front of the monitor, on the left hand side is the Scan Board, on the right hand side is the Video Board. Adjustments should be made with the power off, exercise caution!

On the Scan Board, there is a Scan Rate Jumper (pins 52-61) which controls the scan rate (different resolutions). This jumper should be in the bottom position (pins 52-59), specifying low resolution (525 lines).

Below it to the left are three potentiometers, the lower one (R126) controls the horizontal hold for low resolution. Other "pots" of interest control pincushion (R65), width (R78), height (R4), v-hold (R2), v-center (R24), h-center (R90).

On the Video Board, there is a Sync Jumper (pins 35-37) which controls the sync rate. This jumper should be in the upper position (pins 35-36).

Further details can be found in the Conrac 7211 Manual.

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 CHANGEBACKGROUNDBORDER both take a shade argument but the shade is interpreted di■erently. 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

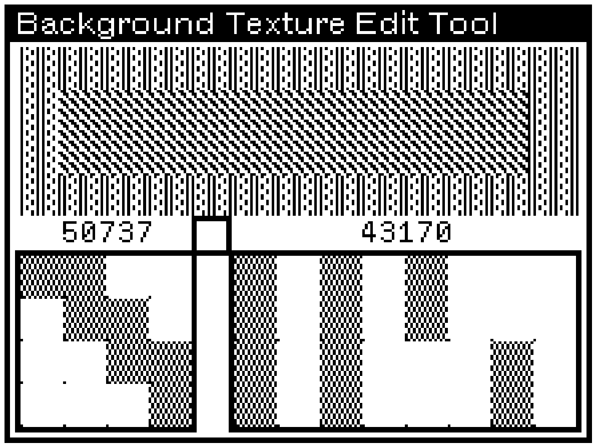
Background Border

■	14	13	12	■	10	9	8
7	6	■	4	3	2	■	0

34850 = 2^15 + 2^11 + 2^5 + 2^1

(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:

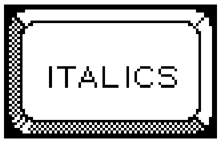


The bottom half of the window has a background texture editor on the left and a border texture editor on the right. The top half of the window shows the background texture within the border texture as it would appear on the screen. Buttoning the small box in the center of the window will change the background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

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 formated 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 the 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]

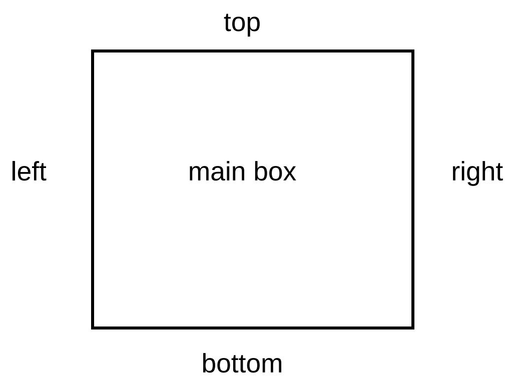
returns non-NIL if *charcode* is allowed in formatstrings.

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

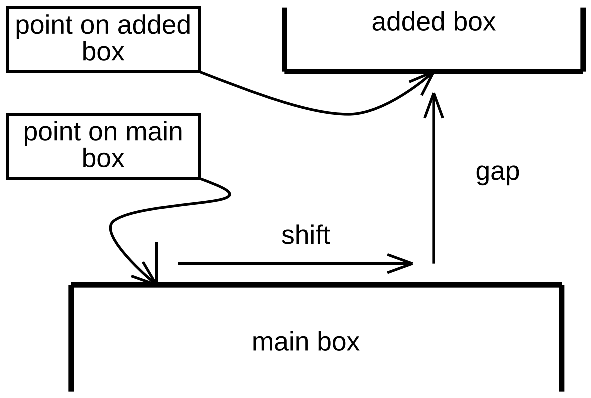
returns non-NIL if *item*'s value is a string (rather than an imageobject) and either *nullOK* is non-NIL or the string is not the nul string.

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).

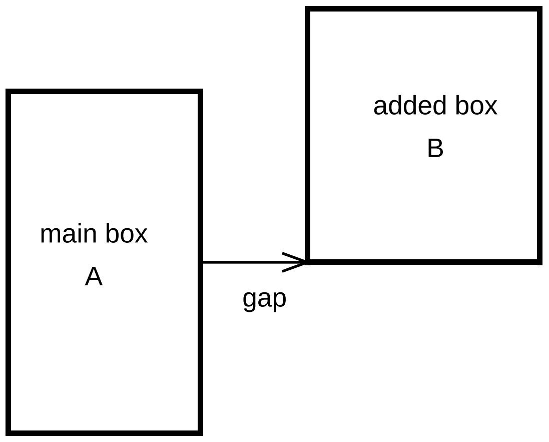
The function returns a list of the form (*region newAddedRegions*) where *region* is the region, w.r.t. the lower left corner of *mainBox*, where *addBox* was placed and *newAddedRegions* is the list of added regions updated to include this newly placed region. If the specified location of *addBox* causes 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*.

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).

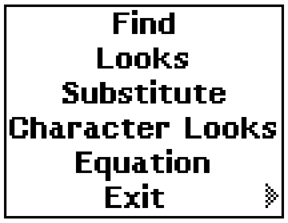
Editing a currently existing equation:

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.

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}`.

BOOTFILECACHTYPES [Variable]

The default types of files that CACHE.BOOT.FILES copies to the BOOTFILECACHEDIRECTORY, initialy `'(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.

EXAMINEDEFS

By Ron Kaplan

This document created in December 2021.

EXAMINEDEFS brings up side-by-side windows for comparison of the definitions of NAME as TYPE from sources SOURCE1 and SOURCE2.

```
(EXAMINEDEFS NAME TYPE SOURCE1 SOURCE2 TITLE1 TITLE2 REGION)                     [Function]
```

The kind of comparison is determined by the value of the variable EXAMINEWITH. If EXAMINEWITH is SEDIT, the different definitions are shown in sided-by-side SEDIT windows. This allows for examination--but not editing--of the definitions: The structures shown in SEDIT are copies of the definitions that GETDEF obtains from the sources and so any changes in either SEDIT will have no lasting effect. A separate SEDIT session with a particular definition (maybe different from either of the sources) can be used in parallell for edits guided by the separate examination. Also, currently, the particular locations of differences in the two definitions are not highlighted by SEDIT selections.

If EXAMINEWITH is COMPARETEXT (the initial value), then the definitions are printed in read-only TEDIT windows (as per PF-TEDIT) and compared with COMPARETEXT. The COMPARETEXT browser makes it easy to iterate from difference to difference.

If SOURCE1 or SOURCE2 is a list and not a GETDEF source specification, then a copy of that structure will be taken as the definition to be shown.

If REGION is a region, it is used as the initial suggestion for the constellation region that side-by-side SEDIT windows will share. (COMPARETEXT windows are determined by the COMPARETEXT protocols.)

TITLE1 and TITLE2 if provided are used to override the default titles of the examination windows. The windows are attached, in that they move, reshape, and close together, but scroll independently.

A second function is provided to bring up side-by-side TEDIT-SEE windows as an aid in examining the differences between files. The optional REGION again specifies a constellation region that the TEDIT's for the two files will share.

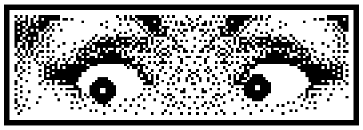
```
(EXAMINEFILES FILE1 FILE2 TITLE1 TITLE2 REGION)                                  [Function]
```

EYECON

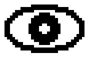
By: Bob Bane (Bane.envos@Xerox.com)

DESCRIPTION

EYECON creates a window with two eyes in it that follow your cursor around. The eyes also wink when you click your mouse buttons: left and right buttons close the corresponding eye, middle button closes both. Here's what it looks like:



The EYECON window can be shrunk into an icon that looks like this:



doing this kills the process which tracks the cursor. Opening the icon restarts the tracking process.

To start it, load EYECON and call (EYECON.OPEN *left bottom*) to create an EYECON window at (*left bottom*).

INSPIRATION

I hacked this up after hearing about something similar that runs under Suntools. The bitmap for the eyes was borrowed from Spy and slightly modified (thanks, Larry!). Julian Orr suggested that the eyes should wink.

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 witht 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 throught 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 accomodate 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 orginal window.or.bm will be restored.

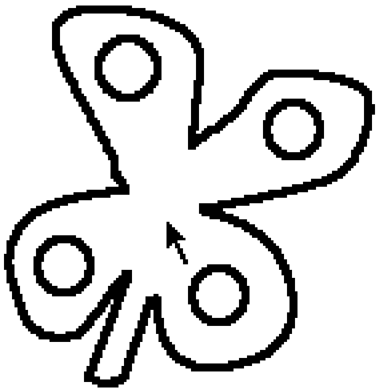
(AUTO.FILL *shade*) [Function]

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

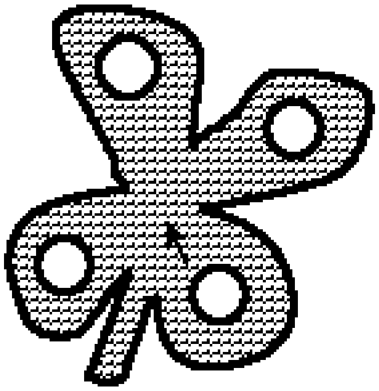
With your mouse pointing inside the appropriate region in a window, this function will fill in the region with the shade specified. This package only works for one bit per pixel bitmaps, color is not supported.

Example:

```
(AUTO.FILL 1234)
```



results in:



Comments and suggetions are welcome.

FINGER

By: Greg Nuyens (mcvax!inria!nuyens@seismo.css.gov)

INTRODUCTION

Finger is a facility for determining and displaying information about other users running Xerox LISP. It displays the user's name, the Etherhostname (or the octal net address when no nameserver is available) and the user's idle time (time since last keystroke or mouseaction). Only other users who have the finger server loaded will be displayed. Users can specify the net radius to query, a list specifying only which users they want displayed, or similarly, only which hosts are to be displayed.

Loading Finger begins the finger server (which responds to queries). To display finger information, the following top-level function is provided:

(FINGER WHO HOST HOPS ICON?) [Function]

WHO is an optional list of usernames specifying which people are to be displayed if a response is received. who defaults to **FINGER.CROWD**, initially **NIL** meaning display all responses.

HOST is an optional list of etherhostnames analogous to *WHO*. Specifying both *WHO* and *HOST* denotes union. *HOST* defaults to **NIL**, denoting all hosts.

HOPS specifies the net radius to query. 0 specifies only nets to which you are directly connected. hops defaults to **FINGER.NET.HOPS**, initially 2.

ICON? specifies whether initial display should be the finger icon

or a display window. *ICON?* defaults to **NIL** meaning display. {typically, in an init file the call would be **(FINGER NIL NIL NIL T)**}.

The display window is updated each time the users bugs the display window with left or middle mouse button, and when most window operations are performed on the display window (shape, repaint, expand from icon, etc.). Right button retains the standard window menu.

Options:

the following are user specifiable variables affecting the operation of Finger.

FINGER.ICON.POSITION a position indicating the original position for the icon. Initially (900,500).

FINGER.DISPLAY.POSITION a position indicating the original position for the display window. Initially (650,325).

FINGER.DISPLAY.HEIGHT height of the display window. Initially 140. The display width is correct for the display format and need not be changed.

FINGER.TIMEOUT milliseconds to wait for the last response packet. Initially 1500.

FINGER.NET.HOPS net radius to be queried.

FINGER.CROWD list of potential users to be displayed (discussed above).

FINGER.INFINITY.MINUTES number of minutes to be considered infinite idle time. Initially 90.

Additional functions of interest to the user are:

(END.FINGER) which kills the finger server process, closes the sockets, closes the windows, etc.

(FINGER.SERVER) will start a finger server process.

FREE MENU CREATOR

By : André Blavier (Rank Xerox France)

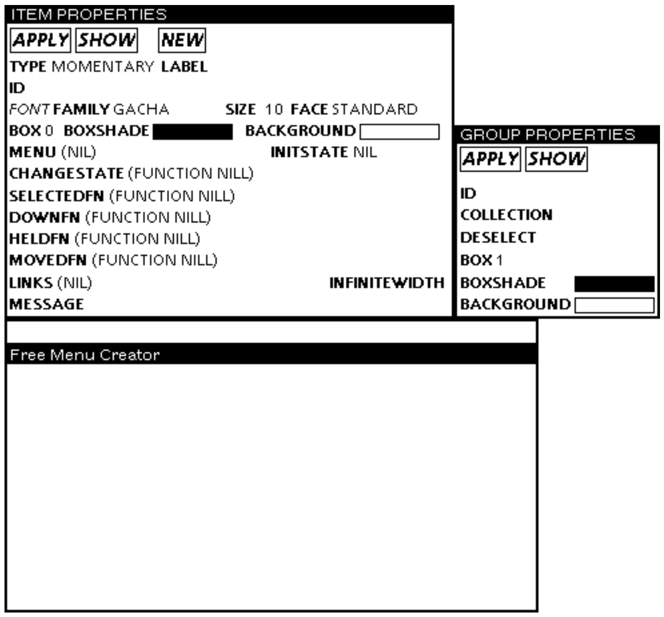
This document last edited on September 29, 1988

The Free Menu Creator application is designed to create interactively Free Menu description lists.

It is a full graphical tool with which you can move, shape, box, group the Free Menu Items and attach properties to them. At any time you can make the application compute the description list and then test the Free Menu that you have just created.

OPENING A FREE MENU CREATOR (FMC) WINDOW

Once you have loaded the application the 'FMCreator' option is added to the background menu. By selecting it you will create a FMC window of the form:



```
ITEM PROPERTIES
APPLY SHOW  NEW
TYPE MOMENTARY LABEL
ID
FONT FAMILY GACHA        SIZE 10 FACE STANDARD
BOX 0 BOXSHADE           BACKGROUND
MENU (NIL)                   INITSTATE NIL
CHANGESTATE (FUNCTION NILL)
SELECTEDFN (FUNCTION NILL)
DOWNFN (FUNCTION NILL)
HELDFN (FUNCTION NILL)
MOVEDFN (FUNCTION NILL)
LINKS (NIL)                     INFINITEWIDTH
MESSAGE

GROUP PROPERTIES
APPLY SHOW
ID
COLLECTION
DESELECT
BOX 1
BOXSHADE
BACKGROUND

Free Menu Creator
```

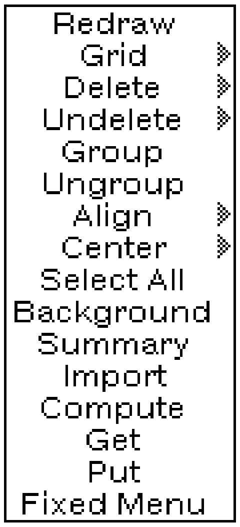
The bottom window is the **main window** : you will add, move ... items to it.

Two menu windows (Free Menus in fact) are attached on top of the main window : the **Item Properties menu** (IP menu) and the **Group Properties menu** (GP menu). They are provided for setting and changing item or group properties.

Between the menus and the main window is the **Prompt window**. Its purpose is mainly to display various information suchas messages, prompts ...

THE RIGHT BUTTON MENU

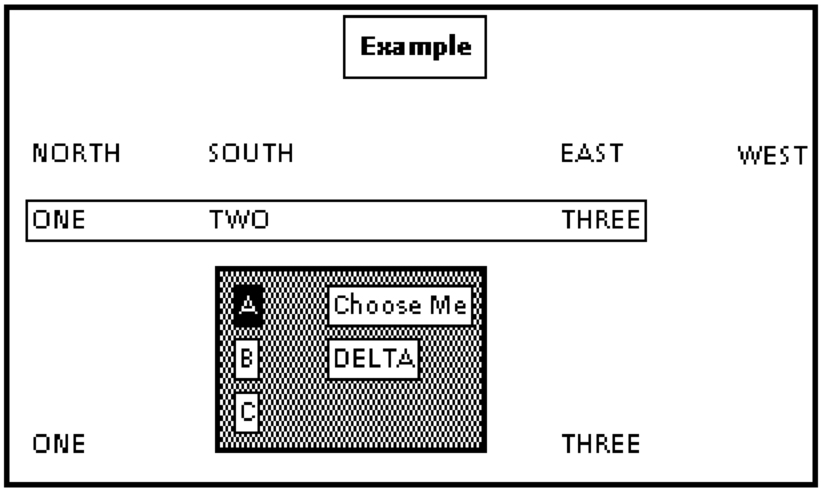
Pressing the right button inside the main window will pop up the following menu:



You can fix this menu on the right edge of the main window by selecting the 'Fixed Menu' option.

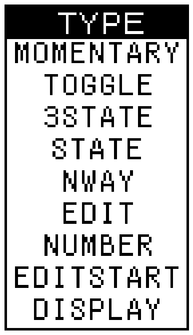
A SAMPLE SESSION

Suppose you want to create the following Free Menu (this example is taken from the FREE MENU Lyric Release Notes):

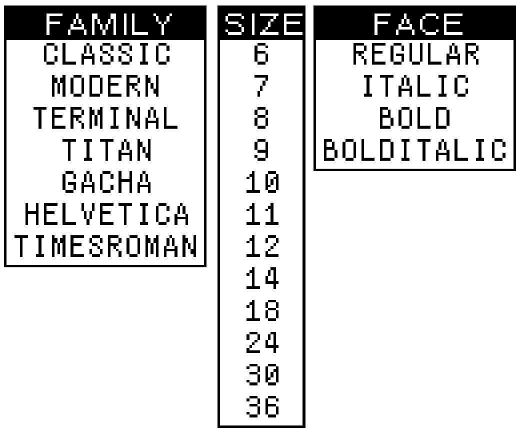


First create the 'Example' item:

- select **TYPE** in the IP menu : the following pop-up menu appears:

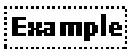


- choose the type of item you want to create, e.g. DISPLAY
- select **LABEL** in the IP menu and type ’Example’
- choose the font by clicking in the **FAMILY**, **SIZE** and **FACE** items, which will cause the following menus to pop up:

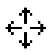


- now, the properties you wanted for that new item are set. So click in the **NEW** field and then move the mouse to the place where you want to put the item.

When you release the mouse button the item is fixed in the window. it is surrounded by a gray rectangle which means that it is **selected**:



If you move the mouse inside the selection rectangle the cursor will change to:

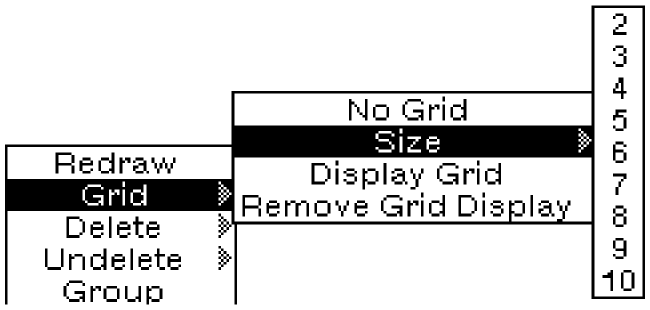


By pressing the left button while the cursor is inside a selection, you can move that selection.

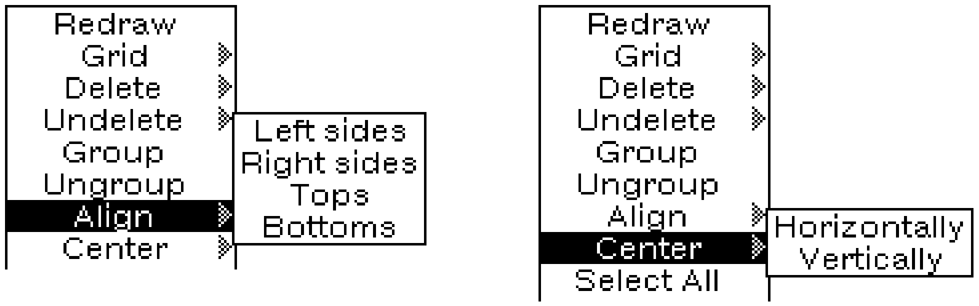
Repeat the same operation for the ’NORTH, ’SOUTH’, ’EAST’, ’WEST’, ’ONE’, ’TWO’ and ’THREE’ items.

To place the items exactly where you want you can use (and combine) the following facilities:

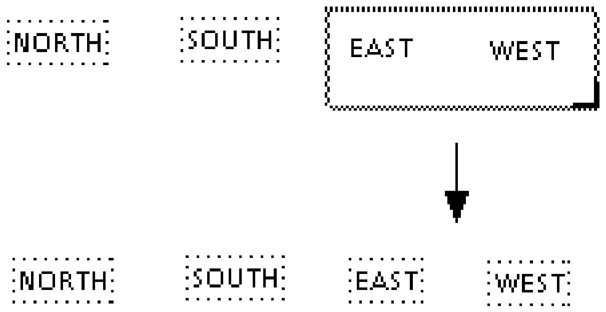
- use the **GRID** (from the right menu):



- align and center a **multiple selection** :

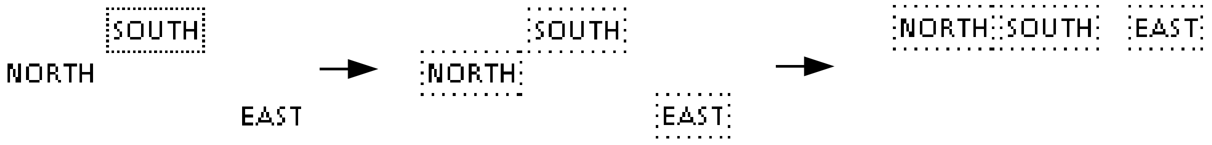


Select multiple items by pressing the left button and the **META** key, while the cursor is above them. You can also extend a multiple selection by pressing Left-Meta outside any item and shaping a ghost region :



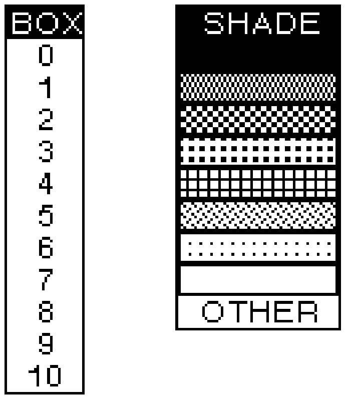
Each item of a multiple selection is surrounded by a dotted rectangle. You can move a multiple selection in the same way you move a unique selection.

The **Align** and **Center** functions all refer to the first selected item. For example, if you choose 'Align Tops' then all the items except for the first one will be moved so that their top is at the same Y coordinate as that of the first item :

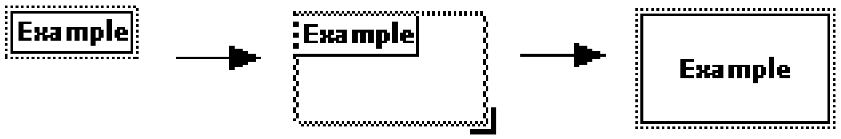


Now, let's box the 'Example' item :
- select the item

- select **SHOW** in the IP menu : this updates the menu according to the item
- set the BOX and BOXSHADE properties :

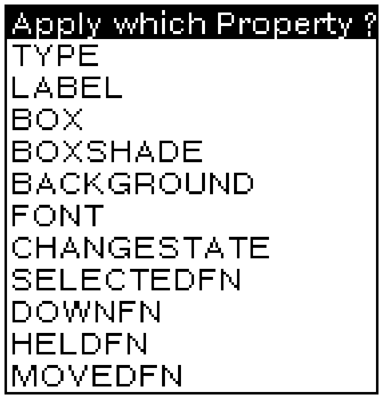


- select **APPLY** : the item is redisplayed surrounded by a box
- press the middle button inside the item : you can now shape the box (the shaping is controlled so that the space between the item and its box is the same horizontally and vertically) :



Whenever you need to change some items's property (ies) , select the item and then select **SHOW** in the IP menu : this updates the menu according to the item. Then, you can change any property, and update the item according to the IP menu by selecting **APPLY**.

You can also change properties of multiple items : make a multiple selection, set the property (ies) you want to change, then select **APPLY** : the following menu pops up :



Only the selected property will be applied to the items.

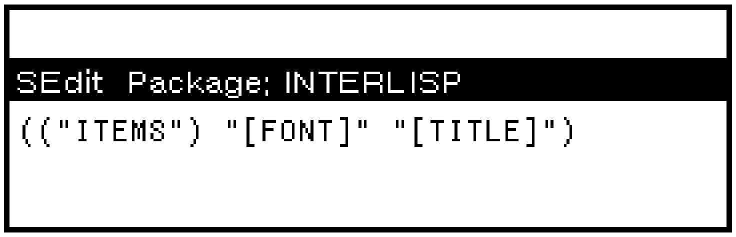
When setting a box property on multiple items, a box is created around each item, not around the whole selection. To wrap a set of items by a box you must first **GROUP** the items. That's what we want to do for the 'ONE', 'TWO', 'THREE' items :

- select the items
- choose **GROUP** from the right menu

Groups are automatically boxed at creation. You can modify groups in the same way as you modify simple items, using the GP menu. Groups can be moved and shaped like simple items.

Now, create the 'A', 'B', 'C' items, setting their TYPE to NWAY. Group the items, select **SHOW** in the GP menu, set COLLECTION to 'COL1', set DESELECT to T - then select **APPLY** : you have created the collection.

Create the 'DELTA' item and then the 'Choose Me' STATE item. For this last item specify a MENU property, selecting **MENU** in the IP menu. This opens the following SEdit window :

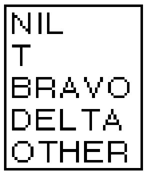


SEdit Package: INTERLISP

```
(("ITEMS") "[FONT]" "[TITLE]")
```

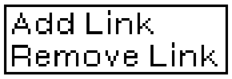
The first element of the list should be an item list, as suitable for standard menus. The second and third elements are optional : FONT should be a list of the form (FAMILY SIZE FACE), TITLE should be a litatom or a string.

Edit the list ((BRAVO DELTA) (MODERN 12 ITALIC)) and close the SEdit window. Now, select **INITSTATE** : this will pop up the following menu :



As you can see the items you just edited are part of this menu. Select DELTA.

One more property is required : the **LINKS** property. Select LINKS : the following menu appears :



Choose 'Add Link' and click on the DELTA item : the link is created.

At any time you can create the actual Free Menu out of your FMC window by selecting the **COMPUTE** option from the right menu. This creates the description list, which is stored in the **FM-DESCRIPTION** global variable. A call to the FREEMENU function is done automatically with FM-DESCRIPTION as an argument, and the Free Menu window is opened.

REFERENCE GUIDE

General window behavior

The main window, the prompt window and the properties windows behave as a whole. Moving, shaping, shrinking and closing can be directed from anyone of them. Shaping affects only the main

window. Closing is protected by a confirmation request when unsaved modifications have occured. The main window is not scrollable.
The right menu, when fixed, can be closed solely.
Hardcopying the main window does not use the standard hardcopy functions. Instead, the contents is 'pretty-hardcopied', but the current version can't print large windows on multiple pages.

The right menu

Redraw : use Redraw to redisplay the contents of the window. This can be useful when items overlap.

Grid : selecting the Grid option displays the grid state in the prompt window.
No Grid : removes the grid.
Size suboptions : specifies a grid size (in pixels). Grid alignment refers to the lower left corner of items.
Display Grid : displays the grid in the window.
Remove Grid Display : removes the grid display, but the grid remains active.

Delete : deletes the selected item(s). Items deleted are saved in a list so they can be undeleted. The number of deleted items is displayed in the prompt window.
Forget save list : deleted items are destroyed and can't be undeleted.

Undelete : undeletes last deleted item.
Last : same effect
All : undeletes all deleted items
List : pops up a menu of all the deleted items. The selected item gets undeleted.

Group : groups a multiple selection. A group can include groups. The components of a group are not individually accesible.

Ungroup : unpacks a group. The ungrouping operation works only at the first level, i.e. included groups are not unpacked.

Align : the Align suboptions all work on a multiple selection (items and/or groups). The alignment operation refers to the first selected item.
Left sides : align left sides of selected items
Right sides : align right sides of selected items
Tops : align tops of selected items
Bottoms : align bottoms of selected items

Center : the Center suboptions all work on a multiple selection (items and/or groups). The centering operation refers to the first selected item.
Horizontally : center items so their center is on the same X coordinate as that of the first selected item
Vertically : center items so their center is on the same Y coordinate as that of the first selected item

Select All : selects all the window contents

Background : pops up a shade menu for setting the window background shade

Summary : creates a TEdit window listing a summary of the window contents. Only interesting properties are described, depending on the item type. Groups contents are indented hierarchically.

Import : allows items importation from a Free Menu. This function works but is currently bugged : importing items with a MENU property is not supported ; when importing groups only their contents are imported.

Compute : generates a description list of the items suitable for the FREEMENU function. The list is stored in the FMC-DESCRIPTION global variable. Opens a Free Menu built out from FMC.

Get : loads the contents of a FMC window, previously stored on disk by a Put operation. The loaded items (groups) are added to the current contents of the window.

Put : saves the contents of a FMC window on disk.

Fixed Menu : when the right menu is poped up by the right mouse button, attaches the menu to the main window. Once the menu is fixed, has no effect.

The Item Properties menu

APPLY : if the selection is unique sets the selected item properties to the properties described in the menu. If the selection is multiple, pops up a menu of properties and sets the selected property of all the selected items according to the menu.

SHOW : updates the menu according to the selected item, thus allowing editing its properties.

NEW : creates a new item which properties are described in the menu.

TYPE : lets the user specify the TYPE property from a pop-up menu.

LABEL : lets the user edit the LABEL property. If no label is edited the item will be displayed in FMC with the pseudo-label ’*NOLABEL*’.
Right-buttoning in this field will start label edition, clearing the field first.
Middle-buttoning puts the cursor in the GETREGION state, and sets the label to the bitmap specified by the user.

ID : lets the user edit the ID property. The edited string is always MKATOMed in the description list.

FONT : FAMILY, SIZE and FACE allow font descriptions from pop-up menus.

BOX : BOX=0 means no box.

BOXSHADE, ***BACKGROUND*** : pop up a shade menu, including an ’OTHER’ option for shade editing.

MENU : opens a SEdit window where the user specifies the MENUITEMS, MENUFONT and MENUTITLE properties. The list must be of the form (MENUITEMS [MENUFONT] [MENUTITLE]).

INITSTATE : pops up a menu of possible INITSTATE values, depending on the item TYPE and possible MENUITEMS.

CHANGESTATE, ***SELECTEDFN***, ***DOWNFN***, ***HELDFN***, ***MOVEDFN*** : open a SEdit window for the corresponding property. The edited list can be of the form (FUNCTION function-name) or a LAMBDA-expression.

LINKS : pops up a menu with 2 items : ’Add Link’ and ’Remove Link’. ’Add Link’ prompts the user to click on the item that is to get linked (this item must have an ID). ’Remove Link’ removes the link.

INFINITEWIDTH : this toggle item is suitable for EDIT or NUMBER items only.

MESSAGE : starts editing the MESSAGE property (right buttoning clears the field first).

The Group Properties menu

APPLY, ***SHOW***, ***ID***, ***BOX***, ***BOXSHADE***, ***BACKGROUND*** behave in the same way as in the IP window.

COLLECTION : starts editing a COLLECTION name. Specifying a COLLECTION property is suitable only for groups which items are NWAY.

DESELECT : a toggle item whose value is relevant only for COLLECTIONs.

About item shaping

The effects of middle buttoning inside an item is different wether the item is boxed or not.
If the item is boxed, shaping will be constrained as described in the sample session, i.e. BOXSPACE must be the same horizontally and vertically.
If the item is not boxed, shaping is constrained so that only the item’s width can be changed, in fact modifying the MAXWIDTH property.

Copy functions

The effects of pressing the COPY key and left buttoning when a FMC window has the TTY depend on what is to be copied :

- when clicking in a SKETCH window, the selected objects are made a graphic LABEL

- when clicking in a FMC window, the selected items are copied in the target window (the source and the target can be the same).

FONTSAMPLER

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

This document last edited on April 27, 1987

INTRODUCTION

The FONTSAMPLER package provides a function which will easily generate sample sheets of your favourite fonts on your favourite printer.

FUNCTIONS

(FontSample *Fonts CharacterSets Printer StreamType*) [Function]

Creates a sample sheet for the font described by the font descriptor(s) *Fonts* on the image stream generated by passing *Printer* and *StreamType* to OPENIMAGESTREAM. *Fonts* may be either a single font descriptor (the result of a call to FONTCREATE) or a list of font descriptors. *CharacterSets* may be either a single number, or a list of numbers representing character sets. The default value of *CharacterSets* (if NIL is passed) is 0 (the Roman Alphabet and Punctuation). FontSample will create a page for each font and character set combination specified, consisting of all the characters of the font/character set combination, arranged on a 16 by 16 grid, and labelled in the default 12pt font for the stream.

For example, to create a sample sheet for the Modern 12pt regular font, character sets 0 and 239 (general and technical symbols 2) on the Interpress printer Fountain one might do:

```
(FontSample (FONTCREATE ’MODERN 12 ’MRR 0 ’INTERPRESS) ’(0 239) ’{LPT}Fountain:)
```

(FontSampleFaked *FontAsList Printer StreamType*) [Function]

There are very often fonts on printers that Interlisp-D cannot speak about, because it is lacking a font width file. FontSampleFaked can produce sample sheets for many of these fonts. *FontAsList* should be a font expressed in the raw form that FONTCREATE takes, *Printer* and *StreamType* are as for FontSample, however passing DISPLAY for *StreamType* will not produce meaningfull results.

To produce a sample sheet for the Vintage 10pt bold font on the Interpress printer Fountain one might incant:

```
(FontSampleFaked ’(Vintage-Printwheel 10 BRR) ’{LPT}Fountain:)
```

NOTES

Be careful if you create sample sheets for many fonts at once – especially the larger type sizes (24pt and up) – you may cause an Interpress printer to hang and require rebooting to recover.

Subject: Interlisp to Commonlisp conversion package
From: darrelj%sm.unisys:COM
To: info-1100%sumex-aim.stanford

FORMACRO and FORMACRO.DFASL -- Still another portable iteration macro
for commonlisp. Its main claims are almost 100% compatibility with
the semantics of the Interlisp-Clisp FOR (especially when used the the
XFORMS which fix a few incompatibilities); and user extensibility
(unfortunately not compatible with IL:I.S.OPR). Embedded keywords
(e.g. IN, COLLECT) may be in any package.

FTPSERVER-MULTI-CONNECTIONS

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

Requires: FTPSERVER, FTPSERVERPATCH and DPUPFTPPATCH

INTRODUCTION

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

INSTALLATION

To install this package, load the FTPSERVERPATCH.LCOM file on the 11xx machine(s) that are to be servers (this will load FTPSERVER if it is not already loaded). Then load the DPUPFTPPATCH.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 FTPSERVERPATCH 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 FTPSERVERPATCH 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 FTPSERVER 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 FTPSERVER 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 `(FTPSERVER)` which is the original FTPSERVER start up), it also will start up a *permanent* FTPSERVER listener on \PUPSOCKET.FTP so regular connection requests can be handled.

ACKNOWLEDGEMENTS

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

GITFNS

By Ron Kaplan

This document was last edited in February 2023.

GITFNS provides a Medley-oriented interface for comparing the files in two different branches of a git repository. This makes it easier to understand what functions or other definitions have changed in a Lisp source file, or what text has changed in a Tedit file. This may be particularly helpful in evaluating the changes in a pull request.

Separately, GITFNS also provides tools and conventions for bridging between git's file-oriented style of development and version control and Medley's residential development style with its own version control conventions. GITFNS allows for intelligent comparisons between Lisp source files, Tedit files, and text files in a local git clone and a local Medley-style working directory, and for migrating files to and from the git clone and the working directory.

Git projects: Connecting git clones to GITFNS capabilities

The GITFNS capabilities operate on pre-existing clones of remote git repositories that have been installed at the end of some path on the local disk. The path to a clone can be used to create a GITFNS "project" for that clone:

```
(GIT-MAKE-PROJECT PROJECTNAME CLONEPATH WORKINGPATH EXCLUSIONS
                  DEFAULTSUBDIRS)                             [Function]
```

where

PROJECTNAME is the name of the project (e.g. MEDLEY, NOTECARDS, LOOPS...)

CLONEPATH specifies the local path to the clone
e.g. {dsk}<users>...>git-medley

WORKINGPATH is optionally the local path to a corresponding Medley-residential working directory (e.g. {dsk}<users>...>working-medley>)

When the project has a WORKINGPATH:

EXCLUSIONS is a list of files and directories to be excluded from comparisons (including what its .gitignore specifies)

DEFAULTSUBDIRS is a list of subdirectories to be use in working-path comparisons when directories are not otherwise specified.

For convenience, if CLONEPATH is NIL or T (and not a path), then a sequence of probes based on PROJECTNAME attempts to find a clone directory (with a .git subdirectory):

```
(UNIX-GETENV PROJECTNAME)    e.g. (UNIX-GETENV 'LOOPS)
```

(UNIX-GETENV (CONCAT PROJECTNAME "DIR") e.g.{UNIX-GETENV 'LOOPSDIR)

(MEDLEYDIR PROJECTNAME)) a subdirectory of MEDLEYDIR

(MEDLEYDIR (CONCAT "../" PROJECTNAME)) a sister of MEDLEYDIR

(MEDLEYDIR (CONCAT "../git-" PROJECTNAME)
(a sister of MEDLEYDIR named git-PROJECTNAME, e.g. git-notecards)

Thus:

If MEDLEYDIR is defined,
(GIT-MAKE-PROJECT 'MEDLEY) will make the MEDLEY project
If NOTECARDS is defined
(GIT-MAKE-PROJECT 'NOTECARDS) will make the NOTECARDS project
If NOTECARDS is not defined but the clone >git-notecards> is a sister of MEDLEYDIR, then the NOTECARDS project will still be created.

If a clone is discovered and a project is created, the value of GIT-MAKE-PROJECT is PROJECTNAME. Otherwise, NIL will be returned if CLONEPATH is T (= no-error), and CLONEPATH=NIL will result in an error.

When they are created, git projects are registered by name on the a-list GIT-PROJECTS, and they can otherwise be referenced by their names.

The variable GIT-DEFAULT-PROJECT, initially MEDLEY, contains the project name used by the commands below when the optional PROJECTNAME argument is not provided.

GIT-MAKE-PROJECT creates a pseudohost {projectname} whose path prefix is the path that resolved to the clone. The file GITFNS in the clone LISPUSERS directory, for example, can be referenced as {MEDLEY}<LISPUSERS>GITFNS.

GIT-MAKE-PROJECT will also create a pseudohost {Wprojectname} for the user's working environment for the project. If WORKINGPATH is provided, that will be the prefix for that pseudohost. If WORKINGPATH is NIL and a directory named working-projectname> is a sister to the clone directory, the pseudohost will point to that.

(GIT-INIT EVENT) [Function]

GIT-INIT creates the default set of projects when GITFNS is loaded, as specified in the variable GIT-DEFAULT-PROJECTS, initially containing MEDLEY NOTECARDS LOOPS TEST. GIT-INIT is added to AROUNDEXITFNS so that new pseudohost bindings for the default projects will be created if the sysout or makesys is started on a new machine.

GIT-DEFAULT-PROJECTS [Variable]

Determines the projects that are created (or recreated) by GIT-INIT. This is initialized for the MEDLEY NOTECARDS LOOPS TEST projects, with CLONEPATH=NIL

GITFNS also defines two directory-connecting commands for conveniently connecting to the git and working pseudohosts of a project:

cdg (projectname) (subdir) [Command]

cdw (projectname) (subdir) [Command]

For example, `cdg notecards library` connects to `{NOTECARDS}/library/`.

Comparing directories and files in different git branches

In its simplest application, `GITFNS` is just an off-to-the-side add-on to whatever work practices the user has developed with respect to a locally installed git project. Its only advantage is to allow for more interpretable git-branch comparisons, especially for pull-request approval. These comparisons are provided by the `prc` ("pull request compare") Medley executive command:

```
prc  (branch)   (DRAFT)  (projectname)                          [Command]
```

This compares the files in `branch` against the files in the main branch of the project (`origin/master` or `origin/main`). Thus, suppose that a pull request has been issued on github for a particular branch, say branch `rmk15` of the default project. Then

```
  prc rmk15
```

brings up a `lispusers/COMPAREDIRECTORIES` browser for the files that currently differ between `origin/rmk15` and `origin/master`. If the selected files are Lisp source files, the Compare item on the file browser menu will show the differences in a `lispusers/COMPARESOURCES` browser. The differences for other file types will be shown in a `lispusers/COMPARETEXT` browser.

If branch is not specified and the shell command `gh` is available, then a menu of open pull-request branches will be provided. If `gh` is not available, the menu will offer all known branches. If the optional `DRAFT` is provided, then the menu will include draft PR's as well as open ones.

If one PR, say `rmk15`, contains all the commits of another (`rmk14`), then the menu will indicate this by

```
      rmk15 > rmk14
```

Note that the `prc` comparison is read-only: any comments, approvals, or merges of the branch must be specified using the normal Medley-external git interfaces and commands.

`prc` is the special case of the more general `bbc` command ("branch-branch compare") for comparing the files in any two branches:

```
bbc  branch1   branch2  (project)                               [Command]
```

This compares the files in `branch1` and `branch2`, for example

```
      bbc rmk15 lmm12   (local)
```

This will compare the files in `origin/rmk15` and `origin/lmm12` in the `GIT-DEFAULT` project. `branch1` defaults to the origin files of the currently checked out branch, the second defaults to `origin/master`. If `local` is non-`NIL`, then a branch that has neither `local/` or `origin/` prepended will default to `local` (e.g. `local/rmk15`) instead of `origin/`. Local refers to the files that are currently in the clone directory, which may not be the same as the origin files, depending on the push/pull status.

Either of the branches can be specified with an atom `LOCAL`, `REMOTE`, or `ORIGIN`, in which case `bbc` will offer menus listing the currently existing branches of that type.

NOTE: Branch comparison makes use of a git command that has a limit (diff.renameLimit) on the number of files that it can successfully compare. A message will be printed if that limit is exceeded, asking whether a larger value for that limit should be applied globally.

The command `cob` ("check out branch") checks out a specified branch:

```
cob  branch  (next-title-string) (project)                       [Command]
```

This checks out branch of project and then executes git pull. The branch parameter may also be a local branch, T (= the current working branch), or NEW/NEXT (= the next working branch). The current working branch is the branch named <initials>nnn, e.g. rmk15. The initials are the value of INITIALS as used for SEDIT time stamps, and nnn is the largest of the integers of all of the branches beginning with those initials.

If branch is NEW or NEXT, then a new initialed branch is created and becomes the user's current branch. Its number is one greater than the largest number of previous initialed branches. If next-title-string is provided, then that string will be appended to the name of the branch, after the initials and next number, and two hyphens. Spaces in next-title-string will also be replaced by hyphens, according to git conventions.

If branch is not provided, a menu of locally available branches pops up.

The currently checked out branch is obtained by the b? command:

```
b?  (project)                                                    [Command]
```

Correlating git source control with separate Medley development

It is generally unsafe to do Medley development by operating with files in a local clone repository. Medley provides a residential development environment that integrates tightly with the local file system. It is important to have consistent access to the source files of the currently running system, especially for files whose contents have been only partially loaded. A git pull or a branch switch that introduces new versions of some files or removes old files altogether can lead to unpredictable disconnects that are hard to recover from. This is true also because development can go on in the same Medley memory image for days if not weeks, so it is important to have explicit control of any file version changes.

GITFNS mitigates the danger by conventions that separate the files in the git clone from the files in the working Medley development directory. The location of the Medley development source tree for a project is given by the WORKINGPATH argument to GIT-MAKE-PROJECT. If WORKINGPATH is T or NIL and there exists a directory >working-projectname> as a sister to the clone, then that is taken to be the WORKINGPATH and thus the prefix for a pseudohost {Wprojectname}.

When Medley development is carried out in the WORKINGPATH, the variable MEDLEYDIR should point initially to the working directory, and the directory search paths (DIRECTORIES, LISPUSERSDIRECTORIES, FONTDIRECTORIES, etc.) all have MEDLEYDIR (or {WMEDLEY}) as a prefix. In that case, the clone for the project, if PROJECTPATH doesn't specify it explicitly, should be located at the >git-medley> sister directory of MEDLEYDIR.

Any back and forth transfer of information between the git clone and Medley development must be done by explicit synchronization actions. Crucially, Medley-updated files do not appear in the clone directories and new clone files do not move to the Medley directories without user intervention.

The files in Medley working tree and the git clone of a project can be compared with the gwc ("git-working-compare") command:

```
gwc  subdirectories (project)                                    [Command]
```

This produces a browser for all the files in the corresponding WORKINGPATH subdirectories that differ from the files in the currently checked out branch of the git clone. If subdirectories is omitted, it defaults to the `DEFAULTSUBDIRS` of the project. If it is `ALL`, then files in all subdirectories that are not found in the project's `EXCLUSIONS` are compared.

In addition to the commands for comparing and viewing files, the menu for this browser also has commands for copying files from the git clone `{projectname}` to `{Wprojectname}` and deleting files from `{Wprojectname}`.

If the master/main branch is the current branch then the menu has no commands to change the clone directory. The browser will show those files that have been updated from a recent merge, and they can individually be copied from the git branch to realign the two source trees with incremented Medley version numbers. If the comparison is with a different branch, say the user's current staging branch, copying files from the working Medley to the git clone or deleting git files will set git up for future commits.

Note that the menu item for deleting Medley files will cause all versions to be removed, not just the latest one, to avoid the possibility that an earlier one is revealed. Deletion for Medley files is also accomplished by renaming to a `{Wprojectname}<deleted>` subdirectory so that they can be recovered if a deletion is in error. Files in the git-clone are removed from the file system immediately, since git provides its own recovery mechanism for those files.

GITFNS does not (yet?) include functions for commits, pushes, or merge for updating the remote repository. Those have to be done outside of Medley through the usual github interfaces, as guided by the information provided by the comparisons.

GRAPHCALLS

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

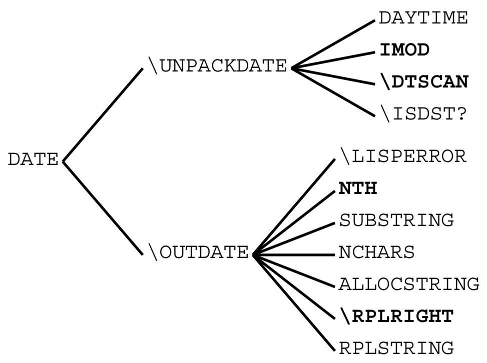
Uses: GRAPHER, MSANALYZE (WHERE-IS & HELPSYS optional)

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:

?=	Print the arguments to the function, if available.	
HELP	Calls HELPSYS on the function.	
FNTYP	Print the function's FNTYP.	
WHERE	Do a WHEREIS (with FILES = T) on the function.	
EDIT	Calls the editor on the function if available for editing.	
TYPEIN	BKSYSBUFs the name of the function into the typein buffer.	
BREAK	Applies BREAK to the function. Its subitems are:	
	BREAKIN	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.
	UNBREAKIN	Undoes BREAKIN.
	UNBREAK	Applies UNBREAK to the function.
	TRACE	Applies TRACE to the function.
	TRACEIN	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.
CCODE	Calls INSPECTCODE on the function if it is compiled code.	
GRAPH	Calls GRAPHCALLS to make a new graph starting with function, inherits the original graph's options.	
FRAME	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:	
	>FRAME	Like FRAME but for all of the functions on the sub-tree starting at the selected node and only for FREEVARS and GLOBALVARS.
	<FRAME	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:

UNBREAK	Does an (UNBREAK), unbreaking all broken functions.
RESET	Resets the counters for the COUNT option and redisplays 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.

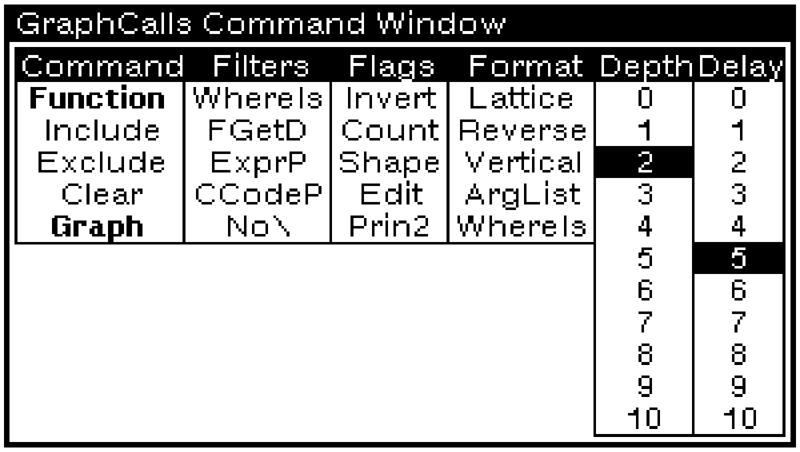
An example of this is `(GRAPHCALLS 'DATE :ADVISE 'INVERT)` and then evaluate `(DATE)`.

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 unrecoverablely.

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



(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.

GraphGroup

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

For internal use only.

The GraphGroup package contains functions for generating a graph showing the structure of a Grapevine distribution list.

DESCRIPTION

There is a single user-callable function:

(GraphGroup *GroupName InfoStream LayoutOptions ExpandNSGroups*) [Function]

GroupName is the group to be graphed.

Because it is rather slow tracking down the whole structure of a group, if *InfoStream* is non-nil GraphGroup will print on *InfoStream* a trace consisting of: a "[" followed by the number of entries in the group each time it starts a (sub-) group, a "." or "?" each time it identifies an individual , and a "]" at the end of a group. T is a good choice for this parameter.

LayoutOptions is a list of options, in property list format, to be passed to the Grapher function LAYOUTSEXPR. Properties recognized are: FORMAT, BOXING, FONT, MOTHERD, PERSONALD, and FAMILYD. See the Grapher documentation for a description of how to use these options. In most cases you will get a satisfactory graph by leaving this parameter NIL. Should you be planning to hardcopy the resulting graph on an NS printer it is reccomended that if you specify a font you use an NS font (such as Classic or Modern) so that the layout spacing is done correctly.

If *ExpandNSGroups* is T, then GraphGroup will attempt to trace into the NSworld when it finds Grapevine RNames that it recognizes as having pseudo-registries that are really NS domain/organizations.

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)

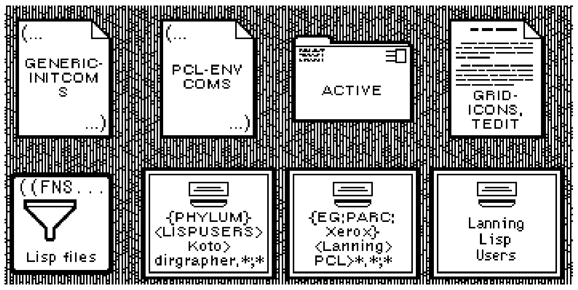
Last edited: September 14, 1987

INTRODUCTION

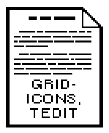
Grid-Icons provides the Lisp user with a set of default window icons that resemble those found in the Viewpoint system. There is an option that the user can set to force these icons to be positioned on a grid, instead of the unrestricted positioning allowed by Lisp.

USING GRID-ICONS

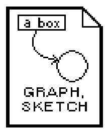
All that is required is loading the file GRID-ICONS. When the file is loaded, it redefines a number of standard window icons in the system.



REDEFINED ICONS

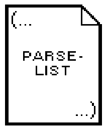


[TEdit icon]

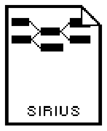


[Sketch icon]

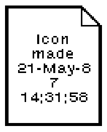
Note that GRID-ICONS not only redefines TEdit and Sketch icons, it also changes the way that the icon title is computed: the host and directory information is removed, so that only the name and extension remain.



[SEdit icon]

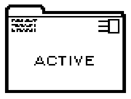


[Loops Browser icon]



[Default icon]

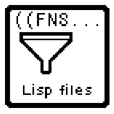
In Lyric, it is possible to redefine the standard icon used by the system. GRID-ICONS uses this, and redefines the standard system icon in LYRIC.



[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 then the original icons, you might want to use a slightly smaller font then 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.

HARDCOPY-RETAIN

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

This document was last edited on December 29, 1987.

Introduction

This module adds a "To a file and a printer" option to the Hardcopy item on the system window and background menus. This option sends the contents of a window or screen region to a printer, and simultaneously creates a file containing a printer-ready copy of it (e.g., an Interpress master); this makes it easy to run off multiple copies later.

Using it

When loaded, HARDCOPY-RETAIN calls the function xcl-user::install-option to side effect il:|WindowMenuItems| and il:|BackgroundMenuItems| and to set il:|WindowMenu| and il:|BackgroundMenu| to nil, thus forcing them to be recalculated.

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 ˆ
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 *HASHARRY* 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.

HELPSYS

By: Doug Cutting (Cutting.PA@Xerox.COM)

additions (CLHS, REPO) by lmm

Uses: DINFO, HASH

The second part of This document last edited on October 7, 1987; now updated August 20.

If given a symbol in the Common Lisp package, the functions below will look up the symbol in the CLHS Index ; these additions work for ? while typing in and the 'man' command.

(CLHS.LOOKUP entry) [Function]

This reads in the CLHS and finds the URL and fragment of the CLHS page there is one. It uses the opener (indicated by CLHS.OPENER) to show the URL if it can.

(CLHS.OPENER) [Function]

This guesses based on your environment what Shell Command can be used to display the CLHS page. One of "wslview" (For Windows with WSL installs), "open" (For Mac OS, "git web--browser" (for various kinds of linux systems or "lynx" (using xterm if available, or CHAT(SHELL) to run lynx. (a fast limited text-only browser). On debian-based systems "sudo apt install lynx" will install it.

CLHS.OPENER [Variable]

If set, will be used; t the opener you prefer from the choices above.

(REPO.LOOKUP keyord) [Function]

In addition, the lookup program has been extended to also look up functions and variables in Library and LispUsers packages, by looking up in the WHEREIS database and, if there is a TEdit or TXT file for that package, opening that. **This only works for functions and variables that are on the same file (according to WHEREIS) as file name of the tedit (or TXT) file, or follow the MAIN-SUBPART convention.**

============================

INTRODUCTION

HelpSys is the interface to the online version of the *Interlisp-D Reference Manual*. It provides both sequential perusal of the manual and random access lookup and display of index entries.

Interlisp-D Reference Manual DInfo Graph

Helpsys uses the **DInfo** library package as a means of accessing the *Interlisp-D Reference Manual*. Once you have loaded Helpsys, selecting **DInfo** from the **Background Menu** will raise a menu which will contain an item named **Interlisp-D Reference Manual**. Selecting this item will start DInfo on the DInfo graph for the manual. (Note: **IRM.HOST&DIR** must be set correctly before HelpSys will work;

see ***Installing HelpSys***, below). See the documentation for DInfo for more information on this package.

?<cr> [EXEC macro]

Helpsys enables the EXEC and TEXEC macro **?<cr>**. Typing this (a question-mark followed by a carriage-return) will cause the CAR of the form currently being typed to be looked up. This works much like the more familiar **?=<cr>** macro which displays arguement lists.

Lookup! [DInfo Command]

Selecting Lookup! from DInfo's menu in the **IRM DInfo Graph** will prompt for and look up a term.

INSTALLING HELPSYS

Helpsys requires a number of files which should be provided with the LispUsers release. These are the 31 chapter files (CHAP*.TEDIT) the top node file (IRMTOP.TEDIT), the file containing the DINFOGRAPH (IRM.DINFOGRAPH) and the index hash file (IRM.HASHFILE).

To install HELPSYS you must copy all these files to one directory, and set the variable IRM.HOST&DIR to the name of this directory.

IRM.HOST&DIR [Variable]

This determines where HelpSys will look for the manual files and hash file. This should be set in your site init file. As file servers can often be quite loaded down, HelpSys will work much faster if you cache these files on the local disk.

IRM.HASHFILE.NAME [Variable]

If this is NIL, HelpSys will look for the hash file as IRM.HASHFILE on IRM.HOST&DIR, otherwise it uses the value of this variable. Note that the hash file *must* be on a random access filing device (eg. the local disk).

PROGRAMMERS' INTERFACE

(IRM.LOOKUP *KEYWORD TYPE GRAPH SMARTFLG*) [Function]

This is the primary function of HelpSys. If *TYPE* is specified, then the primary manual entry for *KEYWORD* of *TYPE,* or the first if there is no primary, will be displayed in *WINDOW.* If *TYPE* is not specified, then a pop-up menu will be raised containing all the manual entries for *KEYWORD*. Primary entries are marked with stars (*'s) on either end of the item. *GRAPH* should be the IRM Dinfo graph, and defaults to IRM.DINFOGRAPH.

(IRM.SMART.LOOKUP *KEYWORD GRAPH*) [Function]

Uses wild card matching if *'s are in *KEYWORD* (* matches any substring) or tries spelling correction. A pop-up menu is raised if more than one wild card match is found. Note that the first time a * appears in a *KEYWORD,* HelpSys will need to load the list of possible keywords for matching against, and only after this list has been loaded will spelling correction be enabled. This is the function called by the Lookup! button in DInfo's FreeMenu. *GRAPH* again defaults to IRM.DINFOGRAPH.

(IRM.RESET) [Function]

Will reset HelpSys so that everything will be reinitialized.

H

By: Roberto Ghislanzoni (Roberto Ghislanzoni:MKT:RXI)

INTRODUCTION

The package H allows the user to augment the Interlisp–D environment with an Horn Clauses Theorem prover: in it it's possible to call semantic attachments (SAs) to Lisp (i.e. lisp functions) as FOL does.

A BRIEF HISTORY

The original idea comes from Chester (Chester,1980), (Chester, 1980): it was revised by Simmons (Simmons, 1984). A prototype of this program was developed at the University of Milan by Vieri Samek Ludovici, Giorgio Tornielli and Roberto Ghislanzoni using VLisp. Currently it is offered on Interlisp–D environment.

USE OF PACKAGE

Load the file H-LOAD.DCOM. This loads all necessary files. On your machine, you have now two environments: the H developer and the H deliver. In H developer you can plan, construct and edit your HKBs (H Knowledge Bases), and use it from Lisp executive window, simply by H.loading the HKBs created. H is a very good logic paradigm for LOOPS: from executive window, it is possible to have as many calls to the prover as you need.

THE H DEVELOPER

From the background menu chose H: this offer you a window in which is active a *read-prove-print* loop. Open as many windows as you want: all HKBs are local to the windows. The H Control Window has the following entries:

— **Show Profile** : This shows in the Prompt Window the current settings for environment bound to the window; the MODE of demonstration (FIRST: stop to the first goal proved; ALL: reach all goals: T: interactive mode); the LIMIT of the search tree; the TRACING of prover: the PMTRACING, that shows the pattern matching at work.

— **Show(Axiom)** : shows the clauses that define a predicate; the submenu allows to see the lambda-definition of a semantic attachment. The choice is made from a pop-up menu.

— **Delete(Axiom)** : this erases from database the clauses choosen by the user. Erases also the SA from the submenu.

— **Edit(Axiom)** : it allows to create and edit both predicates and SA, using the standard DEdit facilities.

— **SetLimit** : sets the limit of the search tree.

— **Mode** : chose the mode of demonstration.

— **Shortform** : it enables or disabled the control for occurrence, so it is not possible to unify the variables already bound in that piece of unification to themselves.

— **Trace** : it enables in a separate window the tracing of demonstration.

— **Trace PM** : it enables the tracing of the unifier.

— **LoadHKB** : loads a H Knowledge base in the environment: the name of KB is shown beside the window; do not provide the .HKB extension to the name: the system does it.

— **SaveHKB** : saves the current environment in a KB: don't provide extension.

— **EraseEnv** : erases the entire environment.

— **Exit** : exits and closes window.

All the windows that H uses ("Show window", "Trace window", "PM Trace window") has the middle button capability in order to allow to dribble into a file everything that is printed in the window; it is very similar to CHAT’s dribble option.

THE H DELIVER

There are a lot of functions available from Lisp Executive that allow the programmer to use the HKBs previously created:

(H.erase) [Function]

Erases all environment (i.e., predicates and SAs) previously loaded.

(H.load *database*) [Function]

Loads H database into environment.

(H.save *database*) [Function]

Saves all current environment into a file

(H.? *pred1 ... predN*) [NLAMBDA Function]

Start the demonstration of the predicates specified. Return the list of predicates proved with the variablesof the call set .

(H.all *variables conjs*) [Function]

Returns the list of all specified variables that satisfy the predicate(s). Remember that variables must begin with a ’:’ (semicolon).

(H.any *howmany variables conj*) [Function]

Returns *howmany* instantiation values of variables that make true the predicate(s).

(H.attach *foo lambda–expression*) [NLAMBDA Function]

Defines a SA named *foo* to be the value of the LAMBDA-expression written as the second argument.

(H.addaxiom *axioms–list*) [Function]

Adds new axioms to the existing one for that predicate.

(H.axiom *axioms–list*) [Function]

Defines new axioms for the predicate. Deletes the previous ones.

(H.del *axiom*) [Function]

Deletes a single axiom from the database; the other axioms for that predicates are not touched.

(H.show) [NLAMBDA Function]

Shows the definition of the given predicates.

(H.the *variable conjs*) [Function]

Returns only one value for the variable that satisfies the goal.

(SET.H.MODE mode num) [Function]

Set the mode of demonstration: mode may be one of atoms ’first, ’all, ’interactive. If atom ’limit, then you must provide the number of depth (default 200).

H PRIMITIVES

Both environments have three important primitives SA:

(set *var expr*)

Sets in the current level of unification the variable to the value of the expression expr.

(assert *axiom*)

Assert in the database the given axiom. without erasing the old ones.

(delete *axiom*)

Delete in the database the given axiom.

In the system also is present the cut facility ('/'), that has similar behaviour as PROLOG cut ('!').

H DEMOS

An example of axioms may be this:

```
(((append () :a :a))
 ((append (:a . :b) :c  (:a . :d)) < (append :b :c :d)))
```

You can call from H-executive:

```
 (append (1 2 3) (4 5 6) :d)
```

that returns

```
 ((append (1 2 3) (4 5 6)  (1 2 3 4 5 6))
```

Also, if you have in your database:

```
(((A 1))
 ((A 2)))
```

and

```
(((B 3))
 ((B 4)))
```

you can call from TTY exec:

```
(H.? (A :l))  --> (((A 1)))
```

or

```
(H.? (A :k) (B :o))  -->  (((A 1) (B 3)))
```

Moreover:

```
(H.all ':k '((A :k)))  --> (2 1)
(H.all '(:a :b) '((A :a) (B :b)))  --> ((2 4) (2 3) (1 4) (1 3))
(H.any 2 ':j '((A :j)))  --> (2 1)
```

For demo. load the HKBs H-MAZE,H- BLOCKS and try the following:

```
(showworld)
```

that shows you the block world situation: try then

```
(please (put the red cube on the blue one))
 (please (pick up cube2))
```

and so on

The other examples solve for you the maze problem and other interesting things: discover by yourself how they work ...

REFERENCES

Chester D., Using HCPRVR, Department of Computer Sciences, University of Texas, Austin, June 1980

Chester D, HCPRVR: a logic program interpreter in Lisp, proc AAAI, Department of Computer Sciences, University of Texas, Austin, June 1980

Simmons R.F.,Computaions from the English, Prentice Hall, New Jersey, 1984

HISTMENU

Original Fugue version By: Danny Bobrow (Bobrow.PA@Xerox.COM)

2020 Medley 3.5 clean sheet reimplementation By: Michele Denber (mdenber@gmail.com)

2022 Medley merge of *Michele Denber implementation* into the *Original Danny Bobrow version* By: Matt Heffron (heffron@alumni.caltech.edu)

INTRODUCTION

HISTMENU is a Xerox Lisp (Medley, Lyric, or Koto) program that provides quick access to commands recently typed in the Exec window. ~~The original HISTMENU was written by Danny Bobrow but seems to have been lost over time. This version was reverse engineered by Michele Denber from a running instance in an old sysout.~~ The source for the Original Danny Bobrow version was located in the archive of Matt Heffron (with a slight? change that cannot be identified). That version is the *basis* of this implementation.

OPERATION

Load HISTMENU.LCOM from your local Lispusers directory. Then call

```
(HistoryIcon [histMenuLength] [histMenuPosition] [histIconPosition])
```

or

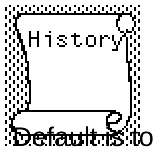
```
(HistoryMenu [histMenuLength] [histMenuPosition])
```

where

`histMenuLength` optionally specifies the number of commands you want displayed. Default is the value of `HistDefaultSlice` which is set to 30 when HISTMENU is loaded.

`histMenuPosition` optionally specifies a position on the display to place the menu. Default is to place it using the mouse.

`histIconPosition` optionally specifies a position on the display to place the graphic icon for the



History Window. Default is to place it using the mouse.

Calling `HistoryMenu` will create the window with only a *standard* window title style icon.

Clicking Left on any menu item will `REDO` that command.

Clicking Middle brings up a pop-up menu that lets you issue one of the four Programmer's Assistant commands `REDO`, `FIX`, `UNDO`, `??` plus an option Delete which removes that item from the History Menu.

Clicking Right bring up a pop-up menu with the three standard options Bury, Move, and Shrink plus an option Update which updates the entries in the History Menu to again show the last n commands from the Exec window.

HistMenu works by using `BKSYSBUF` to "type" the `REDO`, FIX, UNDO, or `??` command with the associated history event number, followed by a `CR`, into the window that currently is accepting input. If there was already something typed in that input, the command would be appended/inserted at the point of the caret and the command will not do what would be expected.

Michele Denber made one change to the original HISTMENU. In the original, it would send the command to whatever window had the keyboard focus. So if you were in a TEdit window, clicking a HISTMENU itwm would place "REDO 354" in the TEdit window. This version only allows commands to go into a selected Exec window. When merging Michele's implementation into the Original version, I (Matt Heffron) disovered that this change makes the History Menu unusable in contexts like a Break or Error window unless it is under an Exec process. To remedy this, I added a global variable `HistMenuExecOnly` which must be non-`NIL` to enable the checking for restricting the imput to an Exec window. `HistMenuExecOnly` defaults to `NIL`.

OPTIONS

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)	Epson America HI-80 (option)
Facit 4551	Western Graphtec MP1000 and FP5301 (option)
Gould Color-writer 6120 and 6320	Houston Instrument DMP-29 (option)
Taxan 710	Nicolet Zeta 8 (option)
IBM 7371	
Roland DYX-880 and 980	(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:

SCALE	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)`.
ROTATE	Paper rotation; value should be 0 or 90, defaults to *landscape* plotting (0).
PAPER	Paper size; value is a small integer, the HP 7475A accepts 3 (A3) or 4 (A4).
TERMINATOR	Label terminator character; value should be a character, the default is '^A'.
VELOCITY	Pen velocity; plotter specific.

IMPLEMENTATION

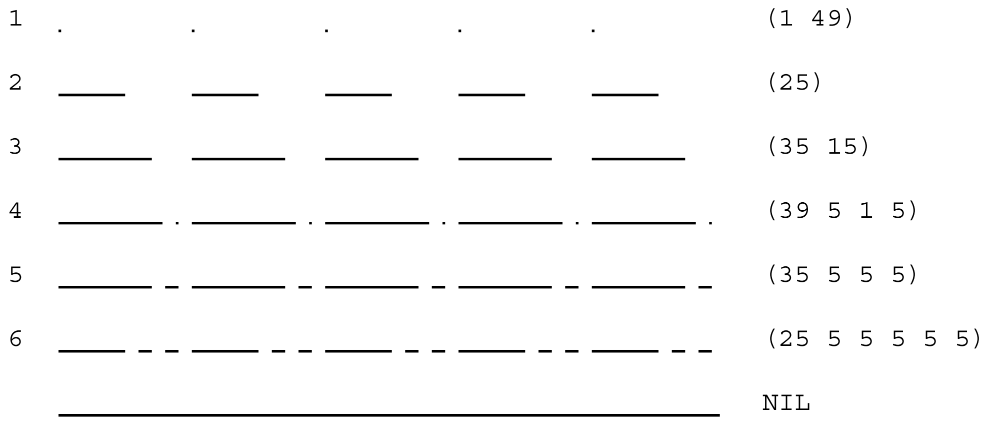
The driver was implemented using an HP 7475A plotter but the plotter output conforms to the more restrictive HP 9872 syntax to be more widely applicable. The driver uses the following variables which may need adjustment for other types of plotters:

HPGL.FONTS	An ALST of font names and (small integer) plotter font numbers.
HPGL.OPTIONS	An ALST of plotter specific options that can be passed to OPENIMAGESTREAM and the corresponding HPGL command to print.
HPGL.DASHING	An ALST of HPGL line types (small integers) and dashing lists.
HPGL.FONT.EXPANSIONS	An ALST of font face expansions (REGULAR, COMPRESSED and EXPANDED) and the relative scale of each.
HPGL.TERMINATOR	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:



If the driver is loaded after SKETCH, the dashing types are added to SKETCH’s dashing menu .

IDEASKETCH

By: Richard Burton (Burton.Pa@Xerox.com)

Uses: Sketch

This document last edited on 26-Jan-89 19:09:56.

INTRODUCTION

Idea Sketch is an adaptation of Sketch that is designed to allow easy jotting down and laying out of ideas. An idea sketch window is actually a sketch window with a command menu tuned for manipulating text and defaults set up for connecting text (i.e. arrowheads are added.)

Selecting the "More Menu" items gets the standard sketch 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:

GraphCalls	If the GRAPHCALLS package is loaded, then calls **GRAPHCALLS** on the current selection.
InspectCode	Opens a new **INSPECTCODE** window on the current selection.
Inspect	Does an **INSPECT** on the value of the current selection. This item has SUBITEMS (see below).
Pretty Print Value	Prompts for region to open a window, and prettyprints the value of the current selection in it. This item has SUBITEMS (see below).
Quit	Closes this window and kills the associated TEDIT process. (Closing the window with the WindowMenu, or by calling **CLOSEW** 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:

Freely	The value of the current selection is determined by any binding that a free-reference from the INSPECTCODE 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.

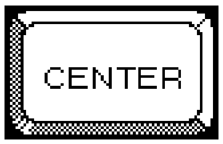
INSPECTCODE-TEDIT also defines the LISPXMACRO **IC** which INSPECTCODE's its argument.

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.

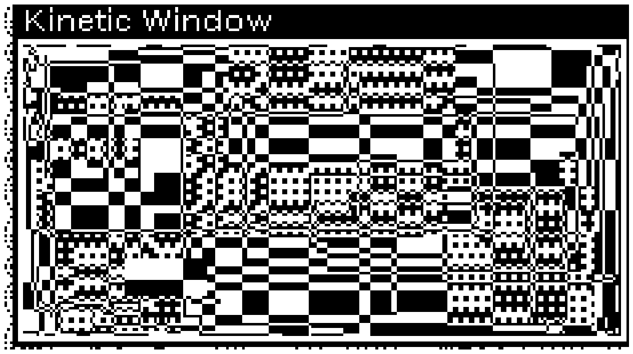
OPERATION

(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.



LAMBDATRAN

By: >>Your Name<< (>>Your net address<<)

>>Other packages necessary to run this one<<

The purpose of this package is to facilitate defining new LAMBDA words in such a way that a variety of other system packages will respond to them appropriately. A LAMBDA word is a word that can appear as the CAR of a function definition, like LAMBDA and NLAMBDA. New LAMBDA words are useful because they enable the user to define his or her own conventions about such things as the interpretation of arguments, and to build in certain defaults about how values are returned. For example, the DECL package defines DLAMBDA as a new LAMBDA word with unconventional arguments such as the following:

```
(DLAMBDA ((A FLOATP) (B FIXP) (RETURNS SMALLP))

(FOO A B))
```

In order for such an expression to be executable and compilable, a mechanism must be provided for translating this expression to an ordinary LAMBDA or NLAMBDA, with the special behavior associated with the arguments built into the function body. The LambdaTran package accomplishes this via an appropriate entry on DWIMUSERFORMS that computes the translation.

Besides executing and compiling, Interlisp applies a number of other operations to function definitions (e.g., breaking, advising), many of which depend on the system being able to determine certain properties of the function, such as the names of its arguments, their number, and the type of the function (EXPR, FEXPR, etc.). The LambdaTran package also provides new definitions for the functions FNTYP, ARGLST, NARGS, and ARGTYPE which can be told how to compute properties for the user's LAMBDA-words.

A new LAMBDA-word is defined in the following way:

1. Add the LAMBDA-word itself (e.g., the atom DLAMBDA) to the list LAMBDASPLST. This suppresses attempts to correct the spelling of the LAMBDA-word.

2. Add an entry for the LAMBDA-word to the association list LAMBDATRANFNS, which is a list of elements of the form: (*LAMBDA-WORD TRANFN FNTYP ARGLIST*), where (*LAMBDA-WORD* is the name of the LAMBDA-word (e.g., DLAMBDA).

TRANFN is a function of one argument that will be called whenever a real definition is needed for the LAMBDA-word definition. Its argument is the LAMBDA-word definition, and its value should be a conventional LAMBDA or NLAMBDA expression which will become the translation of the Lisp LAMBDA-word form. The free variable FAULTFN is bound to the name of the function in which the LAMBDA-word form appeared (or TYPE-IN if the form was typed in).

FNTYP determines the function type of a definition beginning with LAMBDA-WORD. It is consulted if the definition does not already have a translation from which the function type may be deduced. If *FNTYP* is one of the atoms EXPR, FEXPR, EXPR*, or FEXPR*, then all definitions beginning with LAMBDA-word are assumed to have that type. Otherwise, *FNTYP* is a function of one argument that will be applied to the LAMBDA-word definition. Its value should be one of the above four function types.

ARGLIST determines the argument list of the definition if it has not already been translated (if it has, the *ARGLIST* is simply the *ARGLIST* of the translation). It is also a function of one argument, the LAMBDA-word definition, and its value should be the list of arguments for the function (e.g., (A B) in the DLAMBDA example above). If the LAMBDA-word definition is ill formed and the argument list cannot be computed, the function should return T. If an *ARGLIST* entry is not provided in the LAMBDATRANFNS element, then the argument list defaults to the second element of the definition.

As an example, the LAMBDATRANFNS entry for DLAMBDA is (DLAMBDA DECL EXPR DLAMARGLIST), where DECL and DLAMARGLIST are functions of one argument.

Note: if the LAMBDA-word definition has an argument list with argument names appearing either as literal atoms or as the first element of a list, the user should also put the property INFO with value BINDS on the property list of the LAMBDA-word in order to inform DWIMIFY to take notice of the names of the arguments when DWIMIFYing.

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.)

(LifeIdle 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 LifeIdle 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.

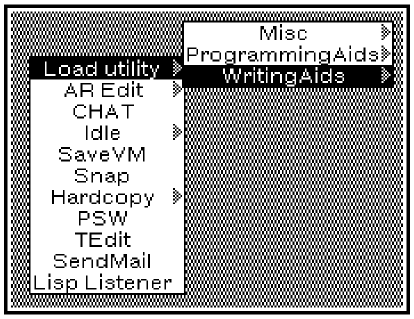
[NOTE: All (advertised) symbols in this utility are in the INTERLISP package.]

EXAMPLE

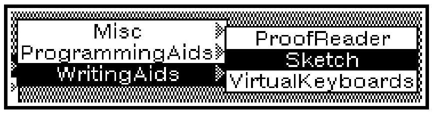
The filepackage command

```
(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 equivilent of selecting the item named *utility-name* from the "Load utility" item on the background menu. If *group-name* is given, only that group undher 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* it 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 20-Dec-1988 00:52:22.

NOTES ABOUT LOGIC MEDLEY RELEASE

This LOGIC release(1.3) is more robust than previous on top of Lyric; there are some bugs fixed. The major enhancement is the possibility of handle multiple SEDIT sessions: there is a global variable, *LOGIC-CLOSE-ON-COMPLETION-FLG* (defaulting to T), that controls the behaviour of the editing: NIL means not to close the editing window and not to perform a check on the correctness of the axiom just typed in, T means to check definitions and to close the editing window.

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 Xerox 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 attachements* (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 specifed theory(es); the user is allowed to dinamically 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 LOGIC-UNIFIER. From within the Xerox 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 LOGIC.LGC.

• (CREATE-VARIABLES): this functions creates and initializes the variables used by the unifier; it takes a few time to perform its job. This call is due to a lack of the Xerox garbage collector: since the unifier uses techniques of redenomination, a great number of symbols is generated; the 1186 does not release the space used by these symbols, and so they fill up the GC table. The workaround is to re-use all the symbols generated.

If you want to port this code on another machine running CL, it is a matter of taste to eliminate this piece of code, with a little hacking on the source codes.

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)

--> ;;creates a window on the screen

(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 tipical example of its use can be the definition of the predicate NOT:
(((not ?formula) :- (wff ?formula) ! (fail))
((not ?formula)))

(TRUE) [Predicate]

This predicate always succeds

(FAIL) [Predicate]

The predicate that never succeds

(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

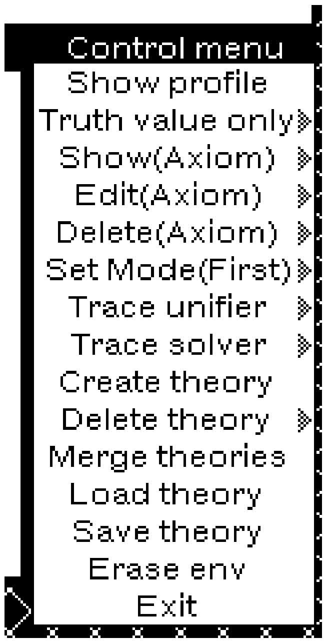
(USE-THEORY ?theory-name) [Predicate]

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 LOGIC- 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 LOGIC-EXAMPLES at the request in the prompt window: mark the theory loaded in the theories window and try:

((APPEND ?A ?B (1 2 3)))

the system will respond you

((APPEND NIL (1 2 3) (1 2 3)))

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:

((APPEND NIL (1 2 3) (1 2 3)))

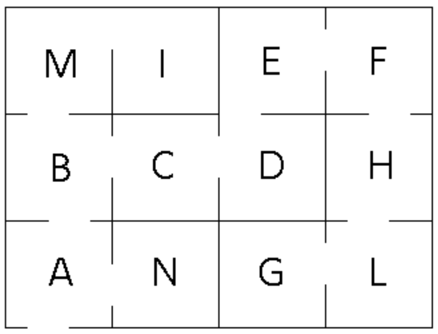
((APPEND (1) (2 3) (1 2 3)))

((APPEND (1 2) (3) (1 2 3)))

((APPEND (1 2 3) NIL (1 2 3)))

NIL

In the theory LOGIC- EXAMPLESa 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’.



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 ’logic-examples)
(prove ’((APPEND ?X ?Y (1 2 3 4)) ’(logic-examples)) --> T
(all ’(?X ?Y)  ’((APPEND ?X ?Y (1 2 3 4)) ’(logic-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!

LOGTIME

Johannes A. G. M. Koomen
Koomen.wbst@Xerox
Koomen@CS.Rochester.edu

October 12, 1989

SUMMARY

LOGTIME is a facility for keeping track of how you spend your time. Clicking on the background menu entry "Log Time" loads a data file and starts up a process. The process wakes at regular intervals and prompts the user for the activity engaged in during the latest interval. The default response is the previous activity. Clicking the right mouse button on the prompt window brings up a menu of known activities. Control-E in the prompt window causes the latest interval to be ignored. Clicking on the background menu entry "Log Time" when a LOGTIME process is already active causes the process to be awoken immediately. Rollout menu entries provide a way to edit the current data, to report on the data accumulated for the current day, and to stop keeping track of time (either "Quit" which updates the datafile, or "Abort" which throws away current data).

DESCRIPTION

(LOGTIME.START *datafile*) [Function]

Starts and returns a new LOGTIME process to keep track of time, using *datafile* as the file to load and store data, unless a LogTime process is already running, in which case NIL is returned. If *datafile* is NIL, it defaults to LOGTIME.DATAFILE (see below).

(LOGTIME.UPDATE) [Function]

Wakes up the LOGTIME process, if any, causing an immediate prompt for an activity.

(LOGTIME.STOP *abortflg*) [Function]

Stops and returns the LOGTIME process currently running, if any. If *abortflg* is NIL, the datafile indicated in the preceeding call to LOGTIME.START is updated.
The "Quit" roll-out entry on the background menu invokes (LOGTIME.STOP).
The "Abort" roll-out entry on the background menu invokes (LOGTIME.STOP T).

(LOGTIME.EDIT *olddatafile newdatafile*) [Function]

Loads *olddatafile* if necessary. Invokes the Lisp editor on a list of entries, where each entry is a list of starting date, ending date and activity, e.g. ("10-Oct-89 13:07" "10-Oct-89 13:12" "Mail").
If all entries can be reparsed upon return from the editor the dataset is modified accordingly. If the data did not come from the LOGTIME process currently running (if any) the data are written back to *newdatafile* if given, otherwise to *olddatafile*.
The "Edit" entry on the background menu invokes (LOGTIME.EDIT).

(LOGTIME.REPORT *bydateflg verboseflg fromdate todate datafile reportfile*) [Function]

Generates a report of the data in *datafile* (defaults to LOGTIME.DATAFILE). If *bydateflg* is non-NIL the report is sorted by date, otherwise by activity. If *verboseflg* is non-NIL, the report lists each single entry in the dataset, otherwise only cumulative time (per date or per activity) is given. If *fromdate* is given (a string acceptable to IDATE) all entries prior to this date are skipped. If the time is omitted from *fromdate* it is assumed to be 0:00. If *todate* is given (a string acceptable to IDATE) all entries past this date are skipped. If the time is omitted from *todate* it is assumed to be 23:59. The report will be generated on *reportfile* (which defaults to LOGTIME.REPORTFILE).
The "Report" entry on the background menu invokes (LOGTIME.REPORT T T <today>).

LOGTIME.INTERVAL [Variable]

The number of minutes the LOGTIME process will sleep in between prompts. Initial value is 15.

LOGTIME.DATAFILE [Variable]

The name of the default datafile. Initial value is (CONCAT (DIRECTORYNAME) "LogTime.Data").

LOGTIME.REPORTFILE [Variable]

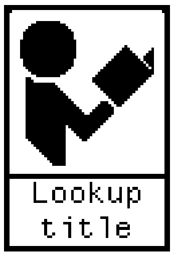
The name of the default reportfile. Initial value is T.

LOGTIME.PROMPT.URGENCY [Variable]

The value used for the PROMPTFORWORD argument *urgency.option*. Initial value is ’TTY.

LOOKUPINFILES

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



INTRODUCTION

The LOOKUPINFILES package is a facility for building quick and easy access to on–line files. It allows search for a target string though 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 iconTitle)

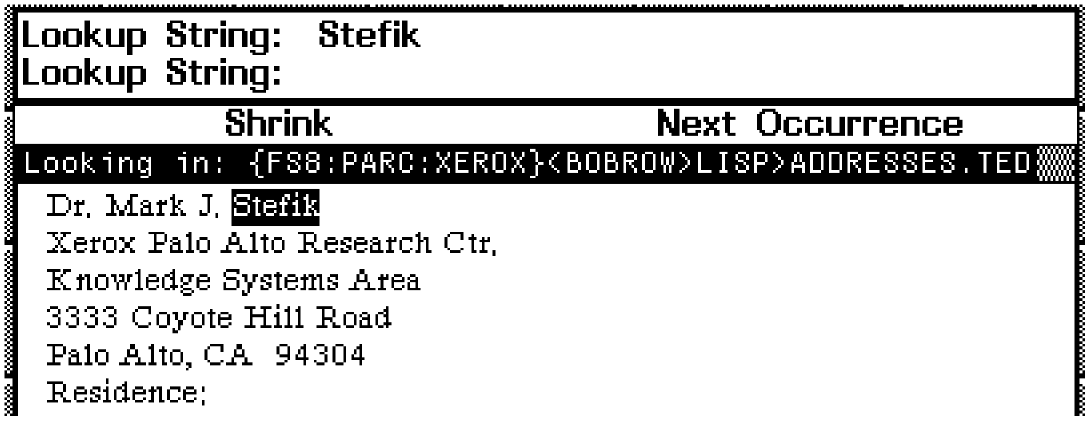
These arguments are used as follows:

fileList	List of file names. Search goes through these files in order
processName	Name appearing in PSW for this lookup process
mainWindowRegion	Region for window showing text found
iconBM	Bit map for icon when mainWindow is shrunk
iconMask	Mask for icon
iconPosition	Position for icon
iconTitle	Title put under icon to distinguish lookup operation

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, ^X, 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 by using the Shrink selection in the title bar .

The window below is taken from the use of this package as an online address book.



```
Lookup String:  Stefik
Lookup String:
Looking in: {FS8:PARC:XEROX}<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 LOOKUPINFILES window

Notes

Caching Files

When you first create 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, click middle button in the title of the main window, and select the option "Edit file named in window title". A TEDIT process editing the file will be set up. This process is independent of the lookup process. To select the file to be edited, rolloff the above item, and select "Select file to edit". A menu of files used by the Lookup process will be presented to you. Selecting one will cause that file to be edited.

To make editing changes visible to the lookup process, PUT the file in TEDIT; when it is done, recache the the file in core. To recache just the file edited, (the one specified in the title bar of the window), select the option "Recache file named in window title" in the middle button title bar menu. You can recache all files by selecting the option "Recache all files" in the title menu (a subselection of the item "Recache file named in window title".

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 beginning of the list of files to be searched, and cached in core.

Deleting a file from the List of Files

To delete from the list of files being used for lookup, select the option "Delete file from list" in the title bar menu. This file will be deleted from the list of files to be searched.

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.SPECIFICAITON [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 a 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 produced 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 produced 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)

Revised by Matt Heffron (heffron@alumni.caltech.edu)

Uses: MASTERSCOPE, FILEBROWSER, DATABASEFNS, and COMMON-MAKE

INTRODUCTION

In its latest incarnation Manager supports MasterScope and improves its performance. It can use DATABASEFNS for managing MasterScope databases, per file. It will use COMMON-MAKE (from Lispusers) to write files in CommonLisp format; and it *borrows* some lower-level functionality from FILEBROWSER.

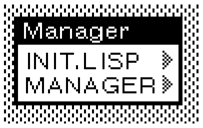
USING MANAGER:

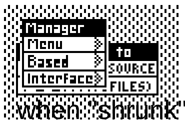
Manager provides a way to perform most common File Manager operations onscreen using menus, both pop-up and permanent. Activity centers around the filelst, or main, menu, and menus of items of a type in the file (like all FNS, or all VARS).

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:

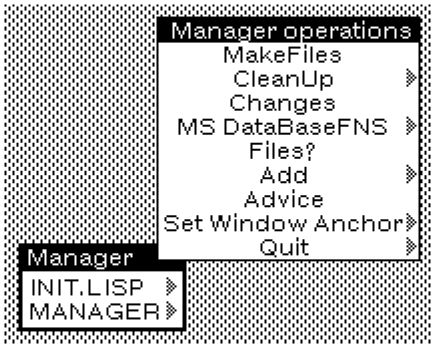


with icon 

when "shrunk".

The names in the FILELST menu can be copy selected.

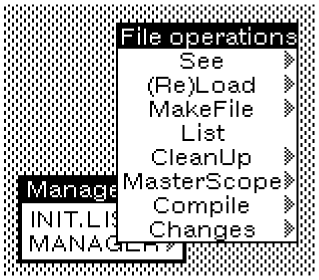
Middle buttoning on the title bar of the FILELST menu pops up a menu of operations which are applied to all loaded files:



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.

List, list the source file on the default printer.

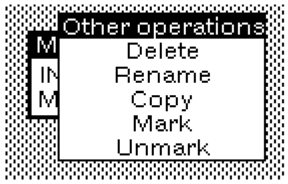
CleanUp:
- CleanUp*, dump the file according to CLEANUPOPTIONS.
- Set default compiler: TCOMPL.
- Set default compiler: CL:COMPILE-FILE.

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:
  - 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.

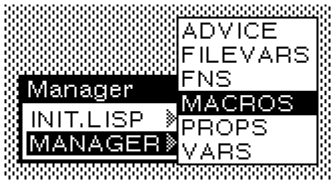
Changes:
- Brief*, prints the changes that have been made to this file.
- Everything, prints the complete list of files changes.
- Edit PL, brings up a lisp editor on the file’s property list.

Middle buttoning on a file in the filelst menu (without sliding off) pops up a menu of generic operations on that file:

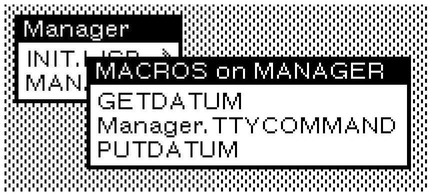


Delete, removes the file object from the system.
Rename, prompts for a new name and renames the file.
Copy, prompts for a new name and copies the file under that name.
Mark, mark the contents of the file as changed.
Unmark, unmark the contents of the file as changed.

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:

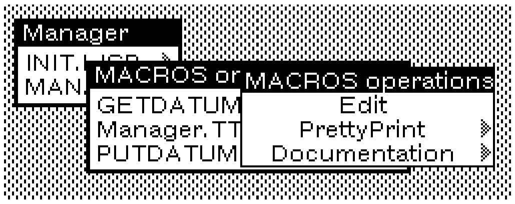


This menu is not pop-up and remains on the display.

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:

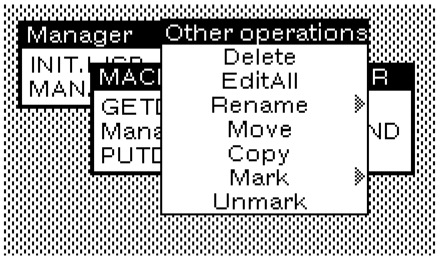


Edit, brings up the source text of the item in a lisp editor.
PrettyPrint:
Show*, prints the source text of the item quickly.
Value, prints the global value of the item’s name (assumed a symbol).
Function Def, prints the global function definition of the item’s name (assumed a symbol).
Property List, 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 occurances 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 readtable argument used to bring up thelisp 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.

Manager.WINDOW-ANCHOR [INITVAR]

This selects which corner of the FILELST menu window remains anchored so that, as files are loaded, its growth will attempt to remain within the Medley screen.
It must be one of ANCHOR-TL, ANCHOR-TR, ANCHOR-BL, or ANCHOR-BR, where ANCHOR-BL is the default. These correspond to the Top Left, Top Right, Bottom Left, and Bottom Right corners, respectively.

Change History

This is a history of edits made to the Manager. Please add your initials and a short description of what you changed to the END of the file. Be sure to include the name of the definition you modified.

andyiii- All menus are sorted now.

andyiii - Appropiate sub-menu update when something is changed that they contain.

andyiii- un-marking a file in the main menu now works and updates all the sub-menus of that file.

andyiii - added option to MAKEFILE menu item for files to write CommonLisp source using common-makefile.

andyiii - added commonlisp DESCRIBE for items

andyiii - Added a way to add files to the file managers main menu

andyiii - Can edit files property list from CHANGES menu

andyiii - Can now mark a whole file from main menu

andyiii - Can chose between TCOMPL (.LCOM files) and compile-file (.dfasl files) This is awkard since is uses the global variable *default-cleanup-compiler*

andyiii - Can get CommonLisp documentation string and descriptons

andyiii - Can now PrettyPrint a value, function def, or prop list and also show how the item would be written to a file

andyiii - Cleaned up specialized menus for FNS, FUNCTIONS, VARS and PROPS

andyiii - All dialog now goes through the MANAGER ACTIVITY WINDOW

RAF 7/31/87 - Fixed the rename option to not specify a source file, uses the ? search (core then file).

RAF 7/31/87 - Added an "edit all occurances of item’s name" option to file relations menu.

RAF 7/31/87 - Manager.ACTIVEFLG is now a special that is bound by all advice to avoid redundant updates inside of themselves. This is a big speed improvement!

RAF 7/31/87 - Fixed Manager.HASITEM and Manager.HIGHLIGHT to use SASSOC, so that list items in menus get highlighted properly.

RAF 7/31/87 - Middle button on Manager file menu now brings up rename, etc. Used to bring up coms to edit (inconsistent).

RAF 7/31/87 - Main menu flashes if bad button/command is given.

RAF 8/4/87 - MANAGER-ADDTOFILES? now initialized to NIL, reducing redundant updates.

RAF 8/14/87 - In Manager.ALTERMARKING: removed extra code which tracked the files containing updated menus. Removed call to Manager.CHECKFILE. Made call to Manager.MAINUPDATE pass T if the reason for marking was DEFINED or DELETED; these cases also call Manager.COMSOPEN.

RAF 8/15/87 - In Manager.DO.COMMAND: moved binding of ACTIVITY-WINDOW-WAS-SHRUNK into the form eval’ed in the process where references are made. Moved setting of ACTIVITY-WINDOW-WAS-SHRUNK after the spot where its referent ACTIVITY-WINDOW is initialized.

RAF 8/16/87 - Advice for LOAD and LOADFNS now call Manager.CHECKFILE instead of Manager.MAINUPDATE (latter only does highlight updating, former can rebuild main menu). Advice for ADDTOFILES? now doesn't disable manager inside of its advised form, so that the ADDTOCOMS and DELFROMCOMS advice will work.

RAF 8/17/87 - Added Manager.FILELSTCHANGED? (which is tricky, since sorting in the main menu changes its order). Manager.CHECKFILE now tests whether the file being checked is in the main menu. If not the main menu is rebuilt. MANAGER fns disables manager around its call to UPDATEFILES. Manager.GETFILE takes a prompt argument (which is now passed in by Manager.DO.COMMAND).

RAF 8/18/87 - Manager.REMOVE.DUPLICATE.ADVICE now disables the manager when it manipulates the advice (to avoid animating the changes in the menus). The advice on LOAD and LOADFNS now call Manager.REMOVE.DUPLICATE.ADVICE.

RAF 8/20/87 - Fixed Manager.MAKEFILE.ADV to handle atomic cleanup options. Also made the top level Manager.RESET call take Manager.ACTIVEFLG, so that manager stays on when reloaded if it was on already. Manager.REMOVE.DUPLICATE.ADVICE now removes *all* duplicates of the first piece of advice (rather than only the second).

RAF 8/21/87 - Made MANAGER-WINDOWS be an initvar so that Manager.RESET from top level sees the right thing on first startup.

RAF 9/2/87 - Changed the manager shrunken bitmap to something more respectable. Added ADVISE and UNADVISE menu options for the ADVICE definer. Added a "Show all advice in effect" option to the manager main window middle button menu. Changed the messages printed out by Manager.DO.COMMAND to all use printout and lambdafont for highlighting.

RAF 9/3/87 - Added a clause in the startup fns MANAGER which reports when FILELST is empty and manager can't start. Also fixed a bug in where marking a file didn't bold the main menu entry (added an updatefiles in Manager.ALTERMARKING). Also caused the advice on the "redundant" call to (MARKASCHANGED :IN DEFAULT.EDITDEFA0001) to fire when FILELST is being edited (seems it was the only way to call markaschanged in that one case).

RAF 11/18/87 - Changed the call to EDITDEF in Manager.DO.COMMAND to include a :DONTWAIT option. The tracks a change in SEdit for the Mototwn release.

RAF 11/18/87 - Added some type checking to the sort testing function Manager.SORT.COMS so that it doesn't convert its arguments to strings unless they're not LITATOMS. This should make menu generation alot faster.

MTH 10/13/2023 - Added Manager.WINDOW-ANCHOR to anchor the window for keeping growth on screen. Added the ICON for the shrunken FILELST window. Moved previously conditional FILESLOAD operations to be always. Fixed a multiple-occurrence typo in the MasterScope database operations: LOADBFLG should be LOADDBFLG.

MISSILE

By: Anonymous
Maintained by: Frank Shih (Shih.envos@Xerox.com)

INTRODUCTION

MISSILE is a Lisp version of the video game, Missile Command. It was discovered on a file server after a Lisp class.

STARTING MISSILE

Load MISSILE.LCOM, and then call (IL:INIT-MISSILE). Try to destroy incoming missiles by clicking the mouse button in the sky. The rest should be obvious. Warning: the game makes lots of noise.

RESTARTING MISSILE

After a game of MISSILE, it can be restarted from the background menu.

MODERNIZE

By Ron Kaplan

This document was last edited in May 2022

MODERNIZE is a simple Lispusers package that changes the mouse actions on Medley windows so that moving and shaping can be done in a way that approximates the behavior of windows on modern platforms, Mac, Windows, etc. It also adds some meta keys to emulate more conventional behavior.

Thus, for a window that has been created or transformed in this way, you can move the window by left-clicking in the title bar and dragging the window's ghost region. Or you can reshape by clicking in a corner of the title bar or near the bottom of the window to drag out the ghost region by that corner.

The menu behavior for other buttons or buttons clicked in other positions is unchanged.

For bottom corners, "near" means inside the window within `MODERN-WINDOW-MARGIN` (initially 25) pixels above or to the left/right of the corner.

For top corners, "near" means within the title bar and within the margin from the left/right edges. (Windows that don't have a title-bar, like Snap windows, can be set up so that moving can happen by clicking anywhere, and shaping at the top is determined by the margin inside the window region.

The function `MODERNWINDOW.SETUP` establishes the new behavior for classes of windows:

```
(MODERNWINDOW.SETUP ORIGFN MODERNWINDOWFN ANYWHERE TITLEPROPORTION)
```

ORIGFN is either the name of the BUTTONEVENTFN for a class of windows (e.g. `\TEDIT.BUTTONEVENTFN` for Tedit windows) or it is a function that creates windows of a particulate kind (e.g. SNAPW or `ADD-EXEC`).

`MODERNWINDOW.SETUP` moves the definition of `ORIGFN` to the name `(PACK* 'MODERN-ORIG-ORIGFN)`. It then provides a new definition for `ORIGFN` that does the window moving or reshaping for clicks in the triggering locations, and otherwise passes control through to the original definition.

If `ORIGNFN` is a button event function, then `MODERNWINDOWFN` should not be specified. In that case a new definition for ORIGFN is constructed to provide the desired windowing behavior.

Otherwise, if `ORIGFN` is the function that creates windows of a class (e.g. `SNAPW`), then a `MODERNWINDOWFN` should be provided to create such windows (by calling `(PACK* 'MODERN-ORIG-ORIGFN)`). The definition of `MODERNWINDOWFN` replaces the original definition of `ORIGFN`.

TITLEPROPORTION may be a number between 0 and .5. It specifies the proportion from either edge of the title bar where clicks will be interpreted as window operations. For example, if TITLEPROPORTION is .25, then a title click that is up to a quarter of the way from the left or right edge of the window will trigger the moving operation. Clicks in the middle 50% of the title bar will always pass through.

If the flag `ANYWHERE` is non-`NIL`, especially for windows without a title bar, then the moving behavior is triggered by a click anywhere in the window (except the corners).

Because this works by redefining existing functions, it is important that the `MODERNIZE` package be loaded AFTER Tedit and Sedit, if those are not already in the sysout. And it should be called to upgrade the proper functions for other window classes that might later be added.

Provided these capabilities are already loaded, the following window classes are "modernized" when `MODERNIZE` is loaded:

TEDIT
SEDIT
INSPECTOR
SNAP
DEBUGGER
EXEC
TABLEBROWSER
FILEBROWSER
FREEMENU
GRAPHER
PROMPTWINDOW
SPY
TOTOPW

If it is not known or it is inconvenient to systematically upgrade a button function or a window-creation function, the new behavior can be provided after a particular window has been created, by invoking

```
(MODERNWINDOW WINDOW ANYWHERE TITLEPROPORTION)
```

This saves the windows existing BUTTONEVENTFN as a window property PREMODERN-BUTTONEVENTFN, and installs a simple stub function in its place.

If things go awry:

```
(UNMODERN.SETUP ORIGFN
```

is provided to restore the original behavior for windows whose buttonevent function is ORIGFN.

```
(UNMODERNWINDOW WINDOW)
```

restores a modernized window to its original state.

`MODERNIZE` also augments the Tedit keyboard commands, to increase compatibility with modern interfaces:

`Meta,q`	is a synonym for Quit
`Meta,a`	is a synonym for Select All

`Known issues:`

Clicking at the bottom of an `EXEC` window running TTYIN is effective only when the input line is empty.

.

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.

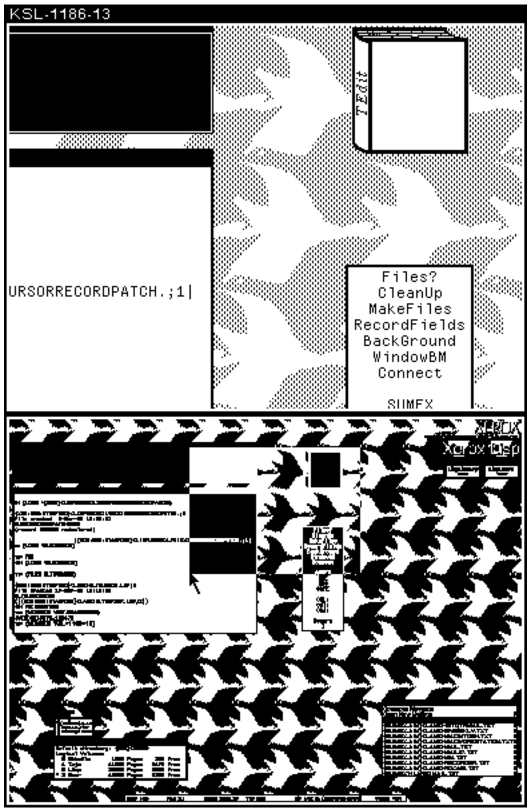
The useful range of scale factors is from **2** to about **8**; a scale factor of **N** will compress the remote screen by **1/N** in width and height and the closeup will cover $\mathbf{1/N^2}$ of the area of the remote screen.

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).


KSL-1186-13
TEdit
URSORRECORDPATCH.;1|
Files?
CleanUp
MakeFiles
RecordFields
BackGround
WindowBM
Connect
SUMEX
XEROX
Xerox Disp

NEATICONS

By: Peter Schachte (quintus!pds@Sun.com)

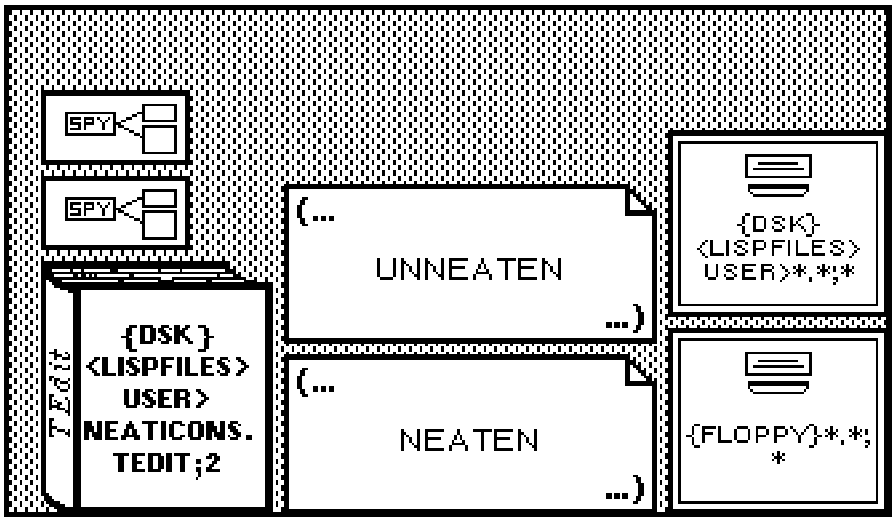
INTRODUCTION

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 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.

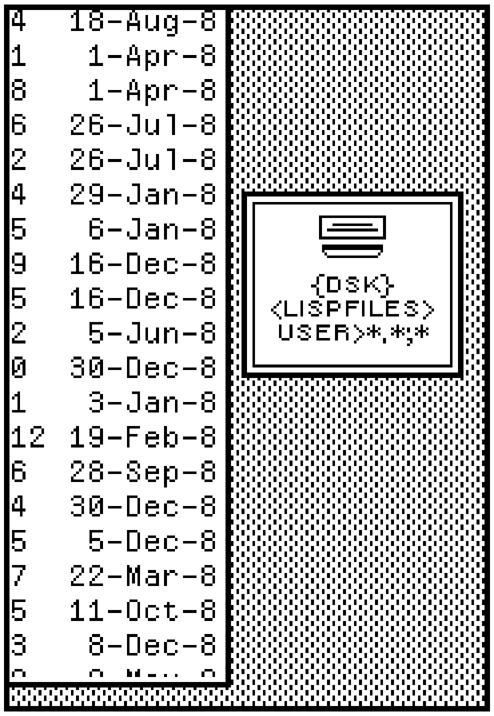
Neat icons line themselves up in a variety of ways. They will flush themselves with the edge of the screen. They will move themselves a fixed number of pixels from the edge of another window. Or they will align one of their edges with the corresponding edge of another window. When you move a neat icon, 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:

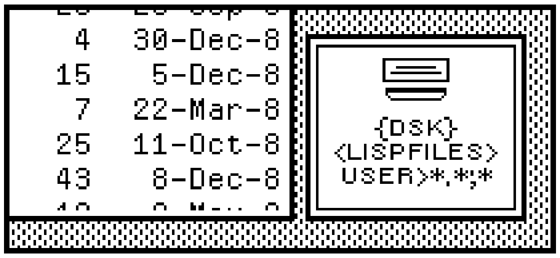


Neat icons can align themselves with the side of a window:



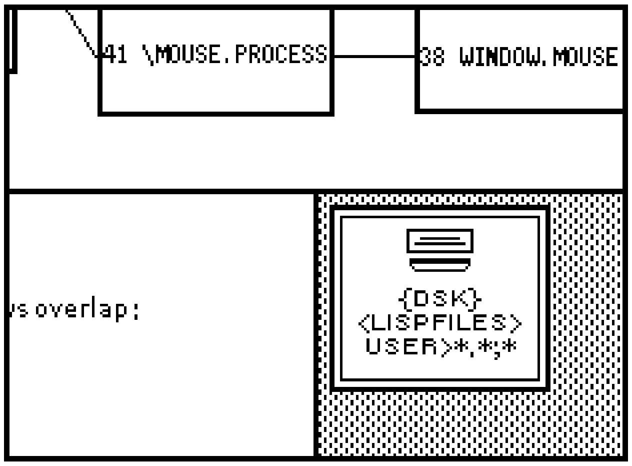
```
4   18-Aug-8
1    1-Apr-8
8    1-Apr-8
6   26-Jul-8
2   26-Jul-8
4   29-Jan-8
5    6-Jan-8
9   16-Dec-8
5   16-Dec-8
2    5-Jun-8
0   30-Dec-8
1    3-Jan-8
12  19-Feb-8
6   28-Sep-8
4   30-Dec-8
5    5-Dec-8
7   22-Mar-8
5   11-Oct-8
3    8-Dec-8
```

But more typically, they will align themselves with the corner of a window:



```
 4   30-Dec-8
15    5-Dec-8
 7   22-Mar-8
25   11-Oct-8
43    8-Dec-8
```

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 &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 promtped 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 possibole to save it on a separate file. The styles are stored as the value of NOTEPAD.STYLES. It includes bitmaps, and msut 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 of 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 of 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).

NOVAFONT

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

INTERNAL

By Nick Briggs
With prodding from Larry Masinter

This utility file allows Lisp to use in fonts in the NOVAFONT format, which is used by Viewpoint.

NovaFont files have in them both the display bitmaps for all sizes of the font, and also the printer widths.

NovaFont files need to be explicitly noticed.

(notice-novafont-file filename) [Function]

After calling notice-novafont-file, the fonts in the given file name will be "known" by the environment. For example,

```
(notice-novafont-file
"{eris}<lispcore>xeroxprivate>fonts>optimamedium.novafont")
```

After this, FONTCREATE will get the bits from the file.

The novafont reader is crafted in such a manner that it only ever reads the file forwards, so you can load fonts from an NS server. For PARC users, note that there are a bunch of NovaFonts (a few malformed...!) on

{starfile public:}<vp applications>*.novafont

including all the "printwheel" fonts, all the Japanese and Chinese character sets (60Q thru 140Q approx), the PC fonts, Quartz .

Unfortunately, it currently doesn't die "gracefully" on the malformed files. It can't just call VP-FONT-P on the files and continue on because it would mean backing up the file pointer.

Instead of loading NovaFonts on demand, all of the display fonts in a NovaFont file can be loaded at once by calling

(load-novafont-file filename) [Function]

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 printedat 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):

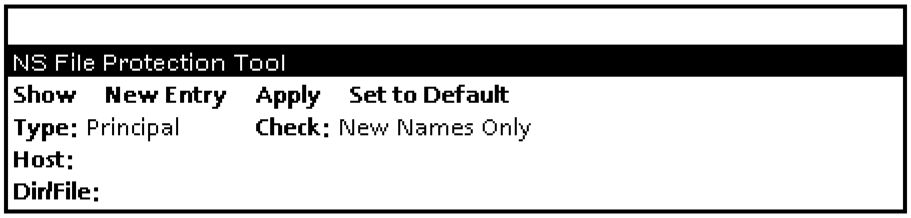
Read	The user may read the file's content and attributes. In the case of a directory, the user may enumerate files in it.
Write	The user may change the file's content and attributes, and may delete the file. In the case of a directory, the user may change the protection of any of the directory's immediate 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
Show **New Entry** **Apply** **Set to Default**
Type: Principal **Check:** New Names Only
Host:
Dir/File:

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.,

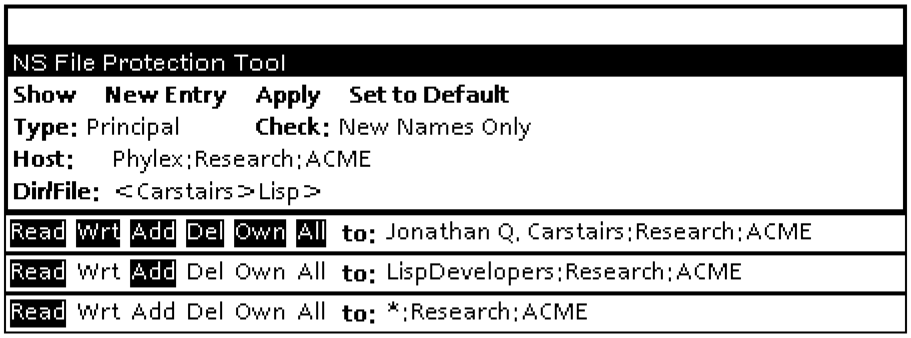
Host: Phylex
Dir/File: Carstairs>Lisp

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

Show **New Entry** **Apply** **Set to Default**

Type: Principal **Check:** New Names Only

Host: Phylex:Research:ACME

Dir/File: <Carstairs>Lisp>

Read Wrt Add Del Own All **to:** Jonathan Q. Carstairs:Research:ACME

Read Wrt Add Del Own All **to:** LispDevelopers:Research:ACME

Read Wrt Add Del Own All **to:** *: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.

OBJECTWINDOW

By Becky Burwell and Ron Kaplan

This document edited on December 21, 2021

An "object window" is a window that contains a sequence of arbitrary image objects arranged either vertically or horizontally. The OBJECTWINDOW package provides the functions for creating such a window, adding objects and manipulating them in various ways, and invoking their IMAGEFNS functions according to mouse or other signals.

An object window is created by the function OBJ.CREATEW:

(OBJ.CREATEW WINDOWTYPE REGION/WINDOW TITLE BORDERSIZE NOOPENFLG SEPDIST BOXFN DISPLAYFN BUTTONINFN) [Function]

The arrangement of objects is determined by the obligatory WINDOWTYPE, either VERTICAL or HORIZONTAL. The other arguments are optional. REGION/WINDOW, TITLE, BORDERSIZE, and NOOPENFLG are passed to CREATEW to create the window. If REGION/WINDOW is a window, it is converted to an object window with TITLE. Otherwise, a new window with region REGION/WINDOW if non-NIL. The objects in the window will be separated (vertically or horizontally) by SEPDIST points, defaulting to 0. The arguments BOXFN, DISPLAYFN, and BUTTONINFN are provided as default functions if is convenient to insert objects whose IMAGEFNS are not fully fleshed out. HARDCOPYFN overrides the default hardcopy function for the window, and HCPYHEADING is used instead of TITLE for hardcopy output.

(OBJWINDOWP WINDOW) [Function]

True if WINDOW is an object window.

(OBJ.ADDTOW WINDOW OBJECT) [Function]

Adds OBJECT at the end of the object sequence in WINDOW.

(OBJ.ADDMANYTOW WINDOW OBJECTS) [Function]

Equivalent to calling OBJ.ADDTOW for each object in OBJECTS.

(OBJ.INSERTOBJECTS WINDOW NEWOBJECTS OLDOBJECT WHERE) [Function]

Inserts NEWOBJECTS at position WHERE (BEFORE or AFTER) with respect to OLDOBJECT.

(OBJ.CLEARW WINDOW) [Function]

Clears the visible objects in WINDOW.

(OBJ.DELFROMW WINDOW OBJECT) [Function]

`OBJECT` is removed from `WINDOW`.

`(OBJ.REPLACE WINDOW OLD.OBJECT NEW.OBJECT DONT.REDISPLAY.FLG)` [Function]

Replaces `OLD.OBJECT` with `NEW.OBJECT` in `WINDOW`, redisplaying the visible objects unless `DONT.REDISPLAY.FLG`.

`(OBJ.FIND.REGION WINDOW SEARCHOBJECT)` [Function]

Returns the region in `WINDOW` occupied by `SEARCHOBJECT`.

`(OBJ.MAP.OBJECTS WINDOW MAPFN)` [Function]

`Applies MAPFN` to each object `OBJ` in `WINDOW`. If `MAPFN` returns an image object, that object replaces `OBJ` in `WINDOW`.

`(OBJ.OBJECTS  WINDOW)` [Function]

Returns the list of objects in `WINDOW`.

PACMAN

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

INTRODUCTION

PACMAN is a Xerox Lisp (Medley, Lyric or Koto) implementation of the arcade game PACMAN. It is fairly faithful to the arcade version, in terms of screen appearance and game dynamics. It runs equally well on Suns, 1132's, 1108's, 1186's, and 1100's (Maikos, Dorados, Dandelions, Doves, and Dolphins). It can run in color for 1132's and 1100's with color boards. Several different methods of user input are supported.

This document describes operational details only. I assume that you know what Pacman is and how to play it.

Load PACMAN.LCOM from your local Lispusers directory. To start the game, type (PACMAN). It will prompt you via menus for input mode, speed, color, and high scores. The menus are described in the next section.

OPTIONS

Input mode: there are currently four ways to control the Pacman, depending on what hardware you have. Although you are asked to pick a mode whenever you start the game, you may switch modes freely at any point during the game.

Mouse - to use this mode, move the mouse in the direction you want the Pacman to go. You only need to move the mouse when you want to change directions.

Keyboard - uses the I, J, K, and space bar as cursor control keys. I is up, J is left, L is right, and space-bar is down. If you look at the arrangement of these keys on the keyboard, you will see that they form an approximate cross. Using the space bar for down (rather than M or ",") lets you control each key comfortably with your right hand; use your thumb for space, and index, middle, and ring fingers for J, I, and L respectively. Note that *you must use the upper-case letters*.

Joystick - this mode lets you connect an Atari joystick to the keyset port on the Alto keyboard of your 1100 or 1132. See the appendix below for details on constructing the interface cable needed. This is the input mode of choice, since the arcade game also uses a joystick. You can, if you want, plug a keyset (if you can find one) into the keyset port, but it is not clear that this offers any particular advantages for data input. I have heard that it is possible to kludge a keyset port on an 1108, but I don't know how to do that.

Voice - perhaps the most exotic input mode, this requires that you have an Interstate Electronics voice recognition board connected to the RS-232 port of your machine. If you possess such a device, call me for specific interfacing details. The commands are "up", "down", "left", and "right". People walking by in the hall will wonder what's going on in your office.

Speed: You can choose one of four speeds for the game. Some effort has gone into making these values similar across different machines, but there's no guarantee they will stay that way. Note that the speed of the game itself will increase (just like the arcade version) as you progress from board to board.

Fast - challenging.

Medium - not unreasonable.

Slow - mellow mode.

Snail - mainly for debugging or the very patient.

Color (y/n): will run the game on your color display (if you have one). The b/w display is blanked out during color mode.

Read high scores (y/n): this option is only available to users on the Xerox Internet, since the high score repository is on this net. If you are an outside or stand-alone user, always answer no to this. The high score list contains the names and score of the top ten players. Note that with the more recent addition of a speed option, this list is somewhat bogus anyway.

Once you have selected your menu options, the game window will appear, along with the message "Insert quarter to start game". You will note a "quarter icon" in the window. Click this to start the game, or click anywhere else to exit. You can skip the introduction screen by clicking the mouse anywhere inside the window while the introduction is in progress.

All of the standard Pacman boards are supported, along with bonus fruit and relative ghost "blue times". You get the correct number of points for eating various objects. On the b/w screen, the ghosts simply invert-video when you eat an energizer; in color they turn blue.

This game is adapted from the author's PDP-11 Fortran version. Call me if you want that one. Please direct all comments, questions, and bug reports to Denber.WBST@Xerox.COM.

KNOWN BUGS

- Pieces of characters are occasionally left lying about the board following collisions.

- Keyboard steering MUST be in uppercase (at least in Medley), lowercase has no effect.

LIMITATIONS

- No sound (that's planned)

- The ghosts do not gain on you as they're chasing you. (You however, can gain on them when you're chasing them).

- The ghosts do not go around turns slower than you do.
- You can’t scare ghosts off by feinting at them.
- Only the first intermission is implemented.
- The top ten score probably no longer works, since it depended on an account on {ICE}.
- Color Pacman does not currently work in the Medley release.
- Mouse steering does not currently work in the Medley release.

APPENDIX: PACMAN CONTROL VIA JOYSTICK

1. You must use a switch-closing type joystick (not a pot type). The Atari 2600 (ie. "Video Computer System") or 400/800 computer joystick works fine. The pinouts given below are for this joystick. Signal descriptions are given too, so you can use other similar devices.

2. You will need a male mouse connector (the pins are embedded in the shell, so it’s hard to the the sex of this type of connector), a 9 pin male D connector, and a short length of six conductor cable.

3. I will use the following convention for pin numbers since the connectors are not clearly labeled. The views below are from the pin-side of the connectors, ie. not the side that the wires are soldered to.

4. I recommend you check the completed assembly for shorts with a VOM and by running KeyTest (available to 1100’s and 1132’s from the NetExec) before trying it out on the game. Shorts or opens won’t hurt the machine, but they will keep the joystick from working properly (obviously).

Keyset connector:

```
-----------------------------------
 \     1   2   3    4   5   6      /
  \ 7   8   9   10  11  12  13  /
   \ 14  15  16  17  18  19    /
    ---------------------------
```

Atari connector:

```
----------------------
\ 1   2   3   4   5 /
 \  6   7   8   9  /
  -----------------
```

Description:

Atari		Keyset	
Pin	Signal	Pin	Signal
2	East	9	PAD1
3	West	10	PAD2
4	South	11	PAD3
5	North	12	PAD4
7	Ground	13	PAD5
9	Fire	14	Ground

Connections:

Atari	Keyset	
2	10	
3	11	
4	12	
5	13	
7	14	
9	9	(not used in PACMAN)

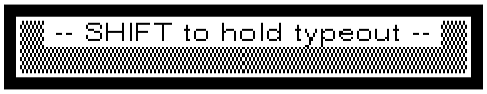
PAGEHOLD

By: Jon L White
Currently maintained by: Bill van Melle (vanMelle.pa@Xerox.com)

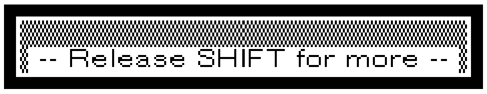
INTRODUCTION

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.

PHONE-DIRECTORY

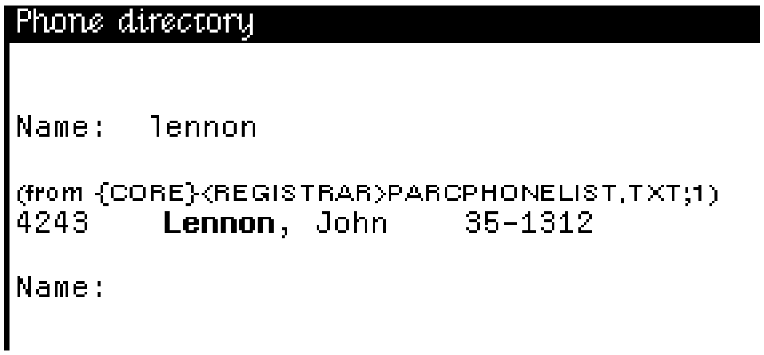
By: smL (Lanning.pa@Xerox.com)

Requires: Lispuser package GREP

INTRODUCTION

The PHONE-DIRECTORY package provides quick and easy access to on–line phone directories.

When you load the PHONE-DIRECTORY package, an icon () will appear on your screen.
Opening this icon will start a directory search program (see example below). To look up a person, type in a name, followed by a return. The PHONE-DIRECTORY program will search the available phone lists and print out those entries that contain the name. This can be repeated as many times as you like. When you are done, just SHRINK the window back to its icon.



```
Phone directory

Name:   lennon

(from {CORE}<REGISTRAR>PARCPHONELIST.TXT;1)
4243    Lennon, John    35-1312

Name:
```

Example PHONE-DIRECTORY window

Variables

PHONELISTFILES [Variable]

PHONELISTFILES is a list of files that contain phone lists. This is usually set in the INIT.LISP file. The PHONELISTFILES should be unformatted files with a single entry per line. Blank lines are permitted. A typical value for PARC users (as defined in PARC-INIT) is

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

Phone-Directory-Pos [Variable]

Phone-Directory-Pos is the initial POSITION for the PHONE-DIRECTORY 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 ←  15
                 YCOORD ← (DIFFERENCE SCREENHEIGHT 75)).
```

This places the icon in the upper left corner of the screen, just above the exec window.

Phone-Directory-Region [Variable]

Phone-Directory-Region is the initial REGION for the PHONE-DIRECTORY window. This is defined as an INITVAR in the file, so you can set it before loading the file. The default value is

```
(CREATEREGION 15 (DIFFERENCE SCREENHEIGHT 258)
              400 250).
```

This places the window in the upper left corner of the screen.

Notes

When you first open the PHONE-DIRECTORY window, the program will copy the PHONELISTFILES to {CORE}, significantly speeding up queries. Bugging in the title of the PHONE-DIRECTORY window with the left or middle mouse button will produce a menu with an option to recache the files from PHONELISTFILES.

PICK

By: Larry Masinter (lmm@iacm.org)
Uses: GITFNS

This document last edited on 10ug 2022

INTRODUCTION

When I get frustrated trying to decide what to work on, I use ’pick’ to choose a project. It has a bunch of different choices.

MODULE EXPLANATIONS

(PICK *option choices)* [Function]

pick *option choice1 choice2 ...* [Command]

Examples

pick	choose something to work on
pick oneof chicken beef tofu pork	choosee one of the named items
pick issue	choose an issue and display it
pick issue 23	if youu give an issue number, just displays it
pick file	choose a file at random: is it where it belongs? can you load it? use it?
pick file lispusers	pick a file on the given (medleydir) directory
pick project	choose a project / repo to work on (from GITFNS GITPROJECTS)
pick file notecards	chooses a notecards repo subdirectoory

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 orderingof 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 MENUS

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 to30 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 and 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 MENUS

A keyword menu appears as a single menu, but is really a data structure encompassing 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 and 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 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 PLOTOBJECT's.

(PLOTCURVE *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 PLOTOBJECT.

(PLOTPOLYGON *plot positions label style menu nodrawflg*) [Function]

As in PLOTCURVE, although a polygon is a closed figure

Returns a POLYGON PLOTOBJECT.

(PLOTTEXT *plot position text label font menu nodrawflg*) [Function]

Text should be a STRING to be printed at position.

Returns a TEXT PLOTOBJECT.

(PLOTFILLEDRECTANGLE *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 FILLEDRECTANGLE PLOTOBJECT.

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.

(PLOTLINE *plot slope constant label style menu nodrawflg*) [Function]

Slope and constant define an analytic line, y = slope * x + constant. If slope is NIL, it is taken to be infinite; i.e. the line is vertical.

Returns a LINE PLOTOBJECT.

(PLOTGRAPH *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 PLOTOBJECT.

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.

(PLOTCOMPOUND *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 accessable via the function COMPOUNDSUBTYPE. Label and menu are as in other plot objects.

Returns a COMPOUND PLOTOBJECT.

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 PLOTOBJECT.

(CREATECURVE *positions label style menu*) [Function]

Returns a CURVE PLOTOBJECT.

(CREATEPOLYGON *positions label style menu*) [Function]

Returns a POLYGON PLOTOBJECT.

(CREATETEXT *position text label font menu*) [Function]

Returns a TEXT PLOTOBJECT.

(CREATEFILLEDRECTANGLE *left bottom width height label texture style menu*) [Function]

Returns a FILLEDRECTANGLE PLOTOBJECT.

(CREATELINE *slope constant label style menu*) [Function]

Returns a LINE PLOTOBJECT.

(CREATGRAPH *graphfn nsamples label style menu*) [Function]

Returns a GRAPH PLOTOBJECT.

(CREATECOMPOUND *compoundtype components label menu*) [Function]

Components must be a list of PLOTOBJECT's.

Returns a COMPOUND PLOTOBJECT.

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.

(ADDPLOTOBJECT *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:

```
(ADDPLOTOBJECT  (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.

(DELETEPLOTOBJECT *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 objected 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 PLOTOBJECTS, PLOTSCALE, SELECTEDOBJECT, PLOTWINDOW, PLOTWINDOWVIEWPORT, PLOTPROMPTWINDOW, 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 'PLOTOBJECTS)
```

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 PLOTOBJECTPROP.

(PLOTOBJECTPROP *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.

(PLOTOBJECTADDPROP *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.

(PLOTOBJECTDELPROP *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 occuring 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 PLOTOBJECT (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 descibes 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 CHOOSETICS is called, which returns an instance of TICINFO. CHOOSETICS 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, CHOOSESCALE 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, CHOOSESCALE 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 tricker example is one in which it is desired to produce unequispaced 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 orginal scale are not equispaced in the log scale, the TICINC field of the returned TICINFO would be a list of the unequispaced 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.

(TRANSLATEPLOTOBJECT *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.

(DRAWPLOTOBJECT *plotobject plot*) [Function]

Draw plotobject in the window asssociated 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.

(ERASEPLOTOBJECT *plotobject plot*) [Function]

APPLY's the plotobject's ERASEFN

(HIGHLIGHTPLOTOBJECT *plotobject plot*) [Function]

Invoked when a plotobject is selected

(LOWLIGHTPLOTOBJECT *plotobject plot*) [Function]

Invoked when a plotobject is deselected

(EXTENTOFPLOTOBJECT *plotobject plot*) [Function]

Computes the extent of plotobject in world coordinates.

Returns an EXTENT, which has fields MAXX, MINX, etc.

(DISTANCETOPLOTOBJECT *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.

(CLOSESTPLOTOBJECT *plot streamposition*) [Function]

Returns the "closest" plotobject on plot's display list to streamposition.

(DESELECTPLOTOBJECT *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.

WHENADDEDFN [Property]

The WHENADDEDFN is called with three arguments, plotobject, plot, and nodrawflg

WHENDELETEDFN [Property]

The WHENDELETEDFN is called with four arguments, plotobject, plot, nodrawflg, and nosaveflg.

WHENDRAWNFN [Property]

The WHENDRAWNFN is called with three arguments, plotobject, viewport and plot.

WHENERASEDFN [Property]

WHENHIGHLIGHTEDFN [Property]

WHENLOWLIGHTEDFN [Property]

WHENTRANSLATEDFN [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.

PLOTBOXPATCH

By: Tad Hogg (Hogg.PA@Xerox.COM)

Uses: PLOT

This document last edited on 26-Jan-89.

INTRODUCTION

This file redefines the plot functions BOXREGION and DRAWPLOT so that each side of the box around the plot need not be drawn. Specifically, giving a plot a non-NIL value for the properties NOLEFT, NORIGHT, NOTOP or NOBOTTOM will eliminate the respective side from the drawing of the plot.

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 in 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 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 PLOTOBJECT 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.

PLOTOBJECT [Datatype]

OBJECTFNS [Field]

Must be an instance of PLOTFNS

OBJECTSUBTYPE [Field]

Describes the plot objects subtype

OBJECTUSERDATA [Field]

Space for a propery list

OBJECTMENU [Field]

The object's MENU

OBJECTLABEL [Field]

Something to print

OBJECTDATA [Field]

Space for a datatype that describes the subtype of this PLOTOBJECT

The field OBJECTFNS must be an instance of PLOTFNS, essentially a vector of functions which implements the generic operations.

PLOTFNS [Datatype]

DRAWFN [Field]

Implements the DRAWOBJECT 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:

(DRAWPLOTOBJECT *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 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.

(ERASEPLOTOBJECT *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

(HIGHLIGHTPLOTOBJECT *object plot*) [Function]

Highlight the object. Used in selection.

(LOWLIGHTPLOTOBJET *object plot*) [Function]

The inverse of HIGHLIGHTOBJECT. With XOR drawing the HIGHLIGHTFN and the LOWLIGHTFN can often be the same.

(MOVEPLOTOBJECT *object dx dy plot*) [Function]

Destructively alter the object's OBJECTDATA, so that its position is moved dx, dy units (in world coordinates).

(LABELPLOTOBJECT *object plot*) [Function]

If it is desired to label the object, the LABELFN will be called. Often the function LABELGENERIC will do the trick.

(EXTENTOFPLOTOBJECT *object plot*) [Function]

Should return an EXTENT, which expresses the range of the object in world coordinates.

EXTENT [Datatype]

MINX [Field]

Minimun extent in the X (horizontal) direction

MAXX [Field]

Maximun extent in the X (horizontal) direction

MINY [Field]

Minimun extent in the Y (vertical) direction

MAXY [Field]

Maximun extent in the Y (vertical) direction

All fields are type floating.

(DISTANCETOPLOTOBJECT *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.

(COPYPLOTOBJECT *object plot*) [Function]

Returns a copy of object. COPYOBJECT will create a new instance of PLOTOBJECT and copyover all the fields of object except for OBJECTDATA. The object’s COPYFN is evoked with the agruments object and plot and is expected to return a new instance of OBJECTDATUM. The objects 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, propname. If the returned value is non-nil it is used as the value for that property on newobject; else the prop value is HCOPYALL’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.

(PRINTPLOTOBJECT *object plot stream*) [Function]

Writes out to stream an HREADable symbolic representation of object. As in COPYOBJECT, PRINTOBJECT takes care of all PLOTOBJECT fields except of OBJECTDATUM. The objects 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 propname 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 fnname) arg) in which case fnname will be invoked at HREAD time with args object plot propname arg and fnname 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:

(

```
(subtypename1  (opname1  function1) (opname2  function2)  ....
(subtypename2  (opname1  function1) (opname2  function2)  ....
 .....
(subtypenamen  (opname1  function1)(opname2  function2) ....
)
```

Creates one instance of PLOTFNS for each subtypename.

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 possitive 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 PLOTOBJECT.

(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 PLOTOBJECT'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 PLOTOBJECT.

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 PLOTOBJECT.

(CREATESAMPLESET *positions constant vertical? side label style menu*) [Function]

Returns a SAMPLESET PLOTOBJECT.

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.

PLOTOBJECTS2

By: Tad Hogg (hogg.PA @ Xerox.com)

Uses: PLOT and PLOTOBJECTS

PLOTOBJECTS2 defines an additional plot object for use with PLOT.

NEW PLOTOBJECT

FILLEDPOLYGON - a polygon with optional shading in its interior.

FUNCTIONS

The following functions provide an add facility for the new object. They are similar to the corresponding functions for the standard plot objects, e.g. PLOTPOINT, etc. The allowed forms of the arguments *texture*, *style*, *menu* and *nodrawflg* are the same as for the standard functions.

(PLOTFILLEDPOLYGON *plot positions label style texture menu nodrawflg*) [Function]

The points in *positions* define a closed polygon. The polygon is filled with *texture*, and the other arguments are the same as for PLOTPOLYGON. If the linewidth specified by *style* is 0, the polygon perimeter will not be drawn.

Returns a FILLEDPOLYGON PLOTOBJECT.

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 creates and returns the new plot object.

(CREATEFILLEDPOLYGON *positions label style texture menu*) [Function]

Returns aa FILLEDPOLYGON PLOTOBJECT.

The actual drawing is done with

(CLILPPED.FILLPOLYGON *clippingregion points texture stream operation windnumber draw? width drawoperation color dashing*) [Function]

which effectively does both of (FILLPOLYGON *points texture stream operation windnumber*) and (if *draw?* is non-NIL)a series of (DRAWLINE *x1 y1 x2 y2 width drawoperation stream color dashing*) for each edge of the polygon except they are clipped against *clippingregion* in stream coordinates.

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 *CHARSET FAMILY ASCIIARRAY*) [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 ASCIITONSTRANSLATIONS. CHARSET is the character set for which to create a translation. FAMILY is the press font family for which ASCIIARRAY is the translation to NS characters. If ASCIIARRAY is not specified, the function looks through all arrays included on ASCIITONSTRANSLATIONS to fill in the translation array.he 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 "Perfector:" %#SIDES 2) FOOBAR)
```

causes the file FOOBAR to be listed two-sided on the print server Perfector: (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 MULTFILEINDEX). 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 reset 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 plist 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 (72 54 504 702)), which on standard letter size paper in portrait mode results in left, bottom, top, and right margins of 1", ¾", ½" and ½", respectively. If the print options passed to LISTFILES call for a two-sided listing, the default region is shifted ¼" to the left. If the print options specify LANDSCAPE mode, the default region is ignored. Any REGION option specified in *PFI-PRINTOPTIONS* must be in points; it is scaled appropriately to the actual hardcopy device being used.

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 functon 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 editting 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.

ACKNOWLEDEGMENTS

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.

PSEUDOHOSTS

By Ron Kaplan

This document was created in January 2022.

A pseudohost identifies the root of a file system that exists as a subdirectory of another pre-existing file system. This gives a shorthand way of operating on a file in the subdirectory of a particular project without having to specify in the name of that file the entire path to its location in a larger file system. For example, suppose that the variable `MEDLEYDIR` contains the path from `{DSK}` to the subdirectory that contains all Medley system files (e.g. `{DSK}<Users>kaplan>Local>medley>`). If not connected to that subdirectory, then the file `COREIO`, say, would have to be reference as

`{DSK}<Users>kaplan>Local>medley>sources>COREIO`.

If `MEDLEY` is defined as a pseudohost with `MEDLEYDIR` as its prefix, then that file can also be identified more succinctly as `{MEDLEY}<sources>COREIO`.

This package implements pseudohost file devices that allow files to be specified and manipulated in this way. The function `PSEUDOHOST` defines a new pseudohost whose files coincide with the files at the end of a prefix directory path:

`(PSEUDOHOST HOST PREFIX)` [Function]

For the Medley example, executing

```
(PSEUDOHOST 'MEDLEY MEDLEYDIR)
```

will set up MEDLEY as a (pseudo) host name that can be used to reference Medley system files. As another example,

```
(PSEUDOHOST 'NC "{DSK}<Users>kaplan>Local>notecards>"
```

would provide a shorthand `{NC}` for accessing the files that make up the Notecards application. Note that the prefix is always interpreted as a directory path, whether or not it ends with an explicit `>` or /.

If `PSEUDOHOST` is called with the name of a previously defined pseudohost but with a different prefix, the new prefix replaces the old. If the prefix is `NIL`, the pseudohost is removed. If `HOST` is a list and `PREFIX` is `NIL`, `HOST` is interpreted as a (host prefix) pair. The target host defaults to `DSK` if `PREFIX` does not have an explicit host.

Of course, full filenames can continue to be used for file access--the pseudohost just provides a systematic abbreviated naming convention. Apart from this convenience, a further advantage is that pseudohost file designators will remain valid if directories are relocated or even moved to different machines, provided that the pseudohost is redefined for the new environment.

Pseudohosts can be used wherever hosts or directories are normally specified. Thus `(CNDIR '{MEDLEY}` or `cd {MEDLEY}` will connect to the underlying `MEDLEYDIR` directory, but the shorter pseudohost will appear in the file and path names returned by the usual system functions `(INFILEP, DIRECTORYNAME, FB *,...)`. When connected to `{MEDLEY}<sources>`, `(INFILEP 'FILEPKG)` will return `{MEDLEY}<sources>FILEPKG`.

When `PSEUDOHOSTS` is loaded, it executes

```
(PSEUDOHOST 'LI LOGINHOSTDIR)
```

so that files in the login directory can always be referenced succinctly with host {LI}, even while connected to another directory.

(PSEUDOHOSTS) [Function]

Returns the (host prefix) pairs of all currently defined pseudohosts.

(PSEUDOHOSTP HOST) [Function]

Returns the (host prefix) pair for a particular pseudohost, NIL if HOST is not a pseudohost.

(TARGETHOST HOST) [Function]

Returns the target host of a particular pseudohost, NIL if HOST is not a pseudohost.

(TRUEFILENAME FILE) [Function]

Returns the name of FILE in its true device, essentially replacing FILE's host by its prefix if it is a pseudohost. Returns FILE (possibly extended with the prefix of the connected directory) if its host is not a pseudohost. FILE may be a stream as well as a name. It may also be a list, in which case the value is the result of applying TRUEFILENAME to each of its elements.

(PSEUDOFILENAME FILE) [Function]

Returns the name of FILE in its pseudo device, if any, essentially replacing FILE's prefix by the hostname of a pseudodevice for that prefix. Returns FILE (possibly extended with the prefix of the connected directory) if it does not match a pseudohost prefix. FILE may be a stream as well as a name. It may also be a list, in which case the value is the result of applying PSEUDOFILENAME to each of its elements.

PS-SEND

By: Matt Heffron (mheffron@orion.cf.uci.edu)

Requires: POSTSCRIPTSTREAM, UNIXCOMM (only on Sun)

The module PS-SEND.LCOM is required by the PostScript ImageStream driver, and will be loaded automatically when POSTSCRIPTSTREAM.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. For Sun Medley users, code has been added to send the completed Postscript files to the local printer (whatever the UNIX environment variable "PRINTER" is set to.)

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 int 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-SKETCH-PATCH

By: Will Snow (snow.envos@xerox.com)

Requires: SKETCH, POSTSCRIPTSTREAM

This module fixes some bugs in SKETCH when interacting with the PostScript driver. This will make the printing of sketches with text in them reasonable. It SYSLOADs SKETCH (from LISPUSERSDIRECTORIES) if it is not already loaded, it does not load POSTSCRIPTSTREAM.

PS-TTY

By: Matt Heffron (mheffron@orion.cf.uci.edu)

Requires: POSTSCRIPTSTREAM, 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 AROUNDEXITFNS 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.

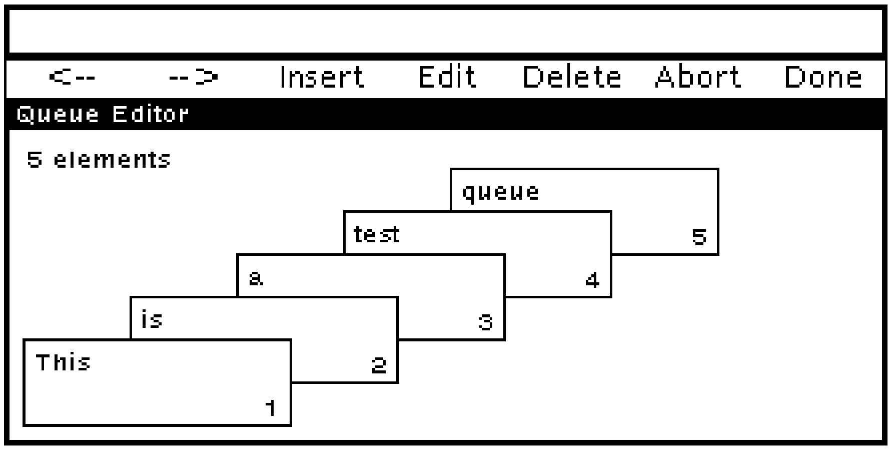
QEdit

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 *PROPS* contained an INSERTFN.

Edit [QEdit command]

Edits the current selection, provided *PROPS* contained an EDITFN.

Delete [QEdit command]

Deletes the current selection, provided *PROPS* contained a DELETEFN.

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 CONTEXT 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 QEDIT.CURRENT.QUEUE (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 NIL 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 NIL or a (possibly new) entry to be used instead of the current selection.

DELETEFN [QEdit property]

If supplied, its value is a function which, when invoked on a queue entry and the user context, returns NIL if the entry should not be deleted. If not supplied, no elements can be deleted. Hint: the function TRUE always returns T.

(QEDIT.CURRENT.QUEUE) [Function]

Invoked from one of the above mentioned functions, returns the queue being edited in its current form. The INSERTFN 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 DELETEFN 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 SMOOTHHIST 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).

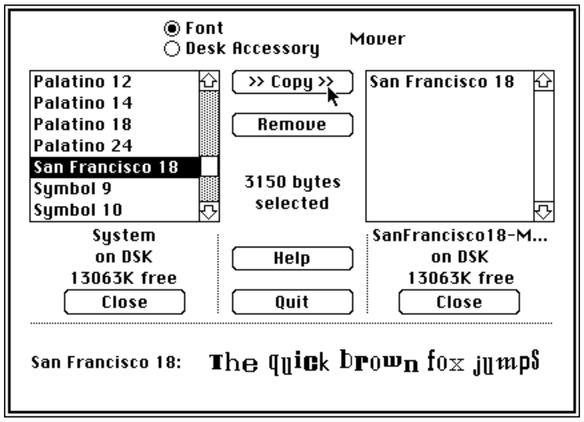
(READAPPLEFONT *STREAM FAMILY SIZE FACE*) [Function]

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:

a
d
)



The individual font files should be moved to Unix AUFS/CAP server (or the equivalent) an 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)

READATATYPE gives @ a read macro de■nition (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 speci■ed, then the read macro returns NIL.

The following form is used in the COMS of the ■le 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)

READISPLAYFONT 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	(*WINDOW POSITION REGION REGIONEVENT*)
REGIONMOVEDFN	(*WINDOW POSITION REGION REGIONEVENT*)
REGIONINFN	(*WINDOW REGION REGIONEVENT*)
REGIONOUTFN	(*WINDOW REGION REGIONEVENT*)
REGIONREPAINTFN	(*WINDOW REGION REGIONEVENT*)
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.

REGIONMANAGER

By Ron Kaplan

This document created in December 2021, last edited September 2023.

Medley comes equipped with a core set of functions for specifying regions and creating the windows that occupy those regions on the screen. But it can be disruptive if not irritating to have to draw out a new ghost region for every invocation of a particular application. Thus the common applications (e.g. TEDIT, SEDIT, DINFO...) implement particular strategies to reduce the number of times that a user has to sweep out a new region. They instead default to regions that were allocated for earlier invocations that are no longer active. TEDIT for example recycles the region of a session that was recently shut down, SEDIT allocates from a list of previous regions, DINFO always uses the same region, but FILEBROWSER always prompts for a new one. Applications that do recycle their regions tend to do so indiscrimately, without regard to the current arrangement of other windows on the screen or the role that those windows may play in higher-level applications.

The REGIONMANAGER package provides simple extensions to the core region and window functions. These are aimed at giving users and application implementors more flexible and systematic control over the specification and reuse of screen regions. It introduces three new notions:

A "typed region" allows the regions of particular applications to be specified, classified, and recycled according to their types.

The size, location, and orientation of a "relative region" is specified with respect to particular screen points and the location of other windows.

A "constellation region" encloses the collection of satellite windows (prompts, menus, etc) that surround the central window of an application.

REGIONMANAGER is innocuous in that explicit user action is required to change the default behavior of any system components.

Typed regions

REGIONMANAGER adds overlay veneers to the core `CREATEW`, `CLOSEW`, and `GETREGION` functions to make it easier to predict and control how different applications arrange their windows on the screen without always needing to respond to a ghost-region prompt.

The `REGION/INITREGION` arguments may now be region-type atoms in addition to either NIL or particular regions as `CREATEW` and `GETREGION` otherwise allow. The type-atom will resolve to a region drawn from a predefined pool of regions associated with that type, if the pool has at least one that is not currently allocated to another window. If the pool has no available regions, then the pool will be enlarged with a region that the user produces from a normal ghost-region prompt, and the type-atom will then resolve to the newly installed region.

A typed-region is marked as "inuse" and therefore unavailable when `CREATEW` assigns it to a window, and the extended CLOSEW marks it as again available when the window is closed. The region of the most recently closed window will be offered the next time a region of its type is requested.

An example of how an application can take advantage of this facility is the TEDIT-PF-SEE package. This provides lightweight alternatives to the PF and SEE commands that print their output to scrollable read-only Tedit windows, specifying PF-TEDIT and SEE-TEDIT as their region types. The user can predefine a preference-ordered sequence of recyclable regions that bring up multiple output windows in a predictable tiled arrangement, without region-prompting for each invocation.

The global variable TYPED-REGIONS is an alist that maintains the relationship between atomic type-names and the list of regions that belong to each type. The list is ordered according to preferences set by the user, and a type-atom is always resolved to the first unused region in its list. If the user is asked to sweep out a new region, that region is added at the end, as the least preferable.   The function SET-TYPED-REGIONS is provided to add or replace TYPED-REGION entries.

(SET-TYPED-REGIONS TYPELISTS REPLACE) [Function]

TYPELISTS is an alist of the form

```
     ((type  . regions )(type  . regions )...)
```

where each regions is a possibly empty list of regions. For convenience, if TYPELISTS is just a literal type-atom, it is interpreted as ((type)), and if it is a list (type . regions) begining with an atom, it is interpreted as ((type . regions). The new regions replace preexisting regions if REPLACE, otherwise they are added at the front.

Typically, a call to SET-TYPED-REGIONS would be placed in a user's INIT file to set up the preference order for the regions that the user wants to participate in this reallocation scheme. If an application uses a type that is not on TYPED-REGIONS, then that type-atom is treated as NIL and always gives rise to the normal ghost-region prompting. Thus a user will observe no change in system behavior if TYPED-REGIONS is left with its initial value NIL. A type that is added with an empty region list (as opposed to not being on the list at all) will allow new regions to accumulate for recycling.

The function REGION-TYPE returns NIL if X is not a typed-region or not a region of type TYPE.

(REGION-TYPE X TYPE) [Function]

In most scenarios the interpretation of a typed region specification is handled automatically by the extended CREATEW and GETREGION functions. Sometimes it may be useful to perform to for the regions dimensions to be entered into other calculations before it is installed in a window. The function GRAB-TYPED-REGION recycles an existing REGION-TYPE window if one meets the optional minimum width and height requirements, otherwise a new region is returned.

(GRAB-TYPED-REGION REGION-TYPE MINWIDTH MINHEIGHT) [Function]

A type can be assigned to an untyped region and installed in a window by the function REGISTER-TYPED-REGION. That region will then be recycled when the window is closed.

(REGISTER-TYPED-REGION REGION REGION-TYPE WINDOW) [Function]

If REGION is NIL, the (presumably) untyped region of WINDOW will be registered. An entry in TYPED-REGIONS will be created for REGION-TYPE if it is not already present.

Relative regions

Two functions are provided to make it easy to create regions relative and oriented with respect to a specified reference point. These may be useful for constructing an application that includes a constellation of windows arranged in a particular relative way.

(RELCREATEREGION WIDTH HEIGHT CORNERX CORNERY REFX REFY ONSCREEN) [Function]

RELCREATEREGION creates a region of dimensions WIDTH and HEIGHT. One of its corners is identified by CORNERX and CORNERY and that corner will be aligned with a reference screen-point determined by REFX and REFY. If ONSCREEN, the WIDTH or HEIGHT will be adjusted with respect to that alignment so that the resulting region is entirely within the screen.

WIDTH and HEIGHT can be given as absolute (natural) numbers or specified relative to the WIDTH and HEIGHT of another region or of the screen. The possibilities are interpreted as follows:

natural number: the number of screen points

list of the form (anchor fraction adjustment), where anchor is a region, window, or an atom SCREEN or TTY. The corresponding dimension of the anchor is mutiplied by `fraction` and `adjustment` is added to the result. For example, specifying `(<window> .5 -1)` results in a WIDTH that is one point smaller than half the width of window's region. `Fraction` and `adjustment` default to 1 and 0 respectively.

region/window/SCREEN/TTY: equivalent to (region/window/SCREEN/TTY 1 0).

CORNERX can be LEFT, RIGHT, or NIL=LEFT, CORNERY can be BOTTOM, TOP, or NIL=BOTTOM. If LEFT/TOP are specified, for example, the region will be displayed down and to the right of the reference point. If RIGHT/BOTTOM, then up and to the left.

The reference-point arguments REFX and REFY are interpreted as follows:

NIL: LASTMOUSEX/LASTMOUSEY

natural number: an absolute screen coordinate

(anchor fraction adjustment) or just region/window/SCREEN/TTY: the quantity determined relative to the size of anchor (as above) is added to the anchors left/bottom produce the REFX/REFY coordinate. In this case, fractions specified as LEFT/BOTTOM/NIL are interpreted as 0 and RIGHT/TOP are interpreted as 1. For example, a specification (<window> .4 -2) for REFY will produce a coordinate 2 points below the level that is 40% of the distance between the bottom and top of the window's region.

For convenience, if REFX is a position and REFY is NIL, then the XCOORD and YCOORD of REFX are taken as absolute values for REFX and REFY.

Also for convenience, if WIDTH is a potentially a list of RELCREATEREGION arguments, then the elements of that list are spread out in a recursive call.

`(RELGETREGION WIDTH HEIGHT CORNERX CORNERY REFX REFY MINSIZE)` [Function]

Calls GETREGION with an initial ghost region as created by RELCREATEREGION. CORNERX and CORNERY determine the ghost region's fixed corner, and the cursor starts at the region's diagonally opposite corner. If MINSIZE is true, then WIDTH and HEIGHT are taken as the minimum sizes of the region, except for adjustments that may be needed to ensure that all corners of the ghost region are initially visible on the screen.

`(RELCREATEPOSITION REFX REFY)` [Function]

Creates a position with X and Y coordinates specified by REFX and REFY references as above.

Constellation regions

Applications are often set up as a constellation of windows, a central or primary window surrounded by some number of satellites for menus, headers, prompts, and secondary outputs. The main panel of a file browser, for example, displays the list of files, but above it are carefully arranged windows for the column headers, summary information, and prompts, and off to the side is the menu of file browser commands. FILEBROWSER interprets the screen region that the user sweeps out for a new browser as the region for the whole constellation,the smallest region that will enclose the central window and all of its satellites. Similarly, the screen region given to TEDIT and SEDIT is divided between the prompt window and the central editing window, again so that the whole constellation (a pair in these cases) fit within the provided region.

Each of these applications is constructed by anticipating the subregions that the satellite windows will occupy after they are attached, decreasing the constellation region by their estimated (using WIDTHIFWINDOW HEIGHTIFWINDOW) or actual sizes, and then using remainder as the region for the central window.

An alternative approach is to construct the central window first, giving it the entire constellation region, and then to have ATTACHWINDOW reshape that window to accomodate the satellite windows as they are attached in sequence. This leads to the same final configuration, but there is no need for separate calculations to pre-adjust the region of the central window.

REGIONMANAGER provides an overlay veneer for ATTACHWINDOW that implements this strategy. If the new argument TAKEFROMCENTRAL is true, then the region of the WINDOWTOATTACH will be substracted from the region of the existing central window according to the EDGE parameter of the attachment.

(ATTACHWINDOW WINDOWTOATTACH MAINWINDOW EDGE POSITIONONEDGE WINDOWCOMACTION TAKEFROMCENTRAL) [Function]

This behavior is also triggered if the UNDERCONSTRUCTION property of the central window is true. Thus, a constellation can be set up by creating all of the satellites and the central window, marking the central window as under construction, and then doing the sequence of attachments. The property can be reset to NIL when the construction is complete, so the central window does not shrink if other windows are attached (e.g. expanded menus) by later user actions.

A somewhat weaker form of a constellation is a collection of windows that are not attached around a central window but stand in a parent-child relationship at least with respect to closing and moving. A parent windows spawns children that respond independently to ordinary window commands (move, shape, close). But the children close when the parent closes, and the children move when the parent moves so that they continue to appear in the same relative positions. These primitives allow the construction of a tree of windows that are dependent in this way.

(CLOSEWITH CHILDREN PARENT [Function]

Establishes a link between the PARENT window and any number of CHILDREN windows such that all CHILDREN will close when PARENT closes. The closing is accomplished by CLOSEWITH.DOIT:

(CLOSEWITH.DOIT PARENT) [Function]

Closes the close-with children of PARENT.

(MOVEWITH CHILDREN PARENT) [Function]

Establishes a link between the PARENT window and any number of CHILDREN windows such that all CHILDREN will move when PARENT closes. The closing is accomplished by MOVEWITH.DOIT:

(MOVEWITH.DOIT PARENT NEWPOS) [Function]

If `NEWPOS` is the new position of `PARENT`, moves each of the move-children so that they stand in the same relation to `PARENT` after it moves as before.

REMOTEPSW

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

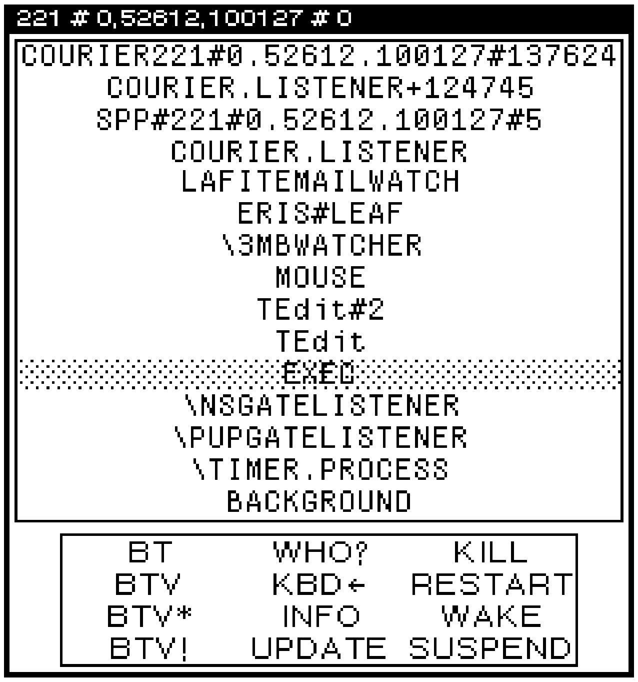
Uses: COURIERSERVE, COURIERDEFS

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
```

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 DEFINITITION

Remote programs are defined via calls to define-remote-program.

define-remote-program	name number version protocol &key :constants [Function] :types :inherits :procedures
	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, remote-procedure-call) 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 - #x1fffffff are defined by Sun, #x20000000 - #x3fffffff 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-S).
constants	a list of pairs (<constant-name> <constant-def>), where <constant-name> is a symbol and a <constant-def> 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 (<type-name> <typedef>) , where a <type-name> is a symbol and a <typedef> 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.

:integer	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.
:unsigned	Lisp: an integer in range 0 to 4,294,967,295 inclusive. XDR: a 4 byte non-negative integer, high order to low order.
:boolean	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.
:hyperinteger	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>)
                                                              (the-rest
                             astructure)))
                             (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 klooge 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 100000 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 il:iphostadress 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 *rpc-socket-cache* and *rpc-well-known-sockets*) or else found by making a call to the Portmapper on the remote machine. If :destsocket 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 create-unix-authentication).
: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-prognum	If you really can't live without it, dynamic-prognum 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, dynamic-version 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 *rpc-msec-until-timeout*.
:msec-between-tries	Number of milliseconds between outgoing UDP packets. Defaults to *rpc-msec-between-tries*.
:errorflg	If :noerrors, ignores remote procedure call errors. If :returnerrors, 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-prognum dynamic-version   [Function]

Returns four values destaddr, socket, program and procedure (Yes, this is real, live multiple value return requiring a multiple-value-bind or something similar.) for consumption by perform-rpc. The arguments to setup-rpc are identical in meaning to the identically named arguments to remote-procedure-call.

open-rpc-stream   protocol destaddr destsocket   [Function]

Returns an rpcstream for use by perform-rpc. Destaddr and destsocket are as returned by setup-rpc and protocol is identical to the protocol argument to remote-procedure-call.

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 arglist credentials protocol &key errorflg leave-stream-open msec-until-timeout msec-between-tries   [Function]

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.

RS232CNetwork

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

Uses: DLRS232C

This document last edited on August 10, 1988.

INTRODUCTION

The RS232C port on a Daybreak can be configured in such a way that it can be used to communicate with a Pup Gateway over a phone line to provide a network communications path equivalent in all but speed to an Ethernet connection.

USE

Load the RS232CNetwork module. It redefines a number of functions, the one of general interest being

(RS232C.INIT *BAUDRATE BITSPERSERIALCHAR PARITY NOOFSTOPBITS FLOWCONTROL LINETYPE*) [Function]

RS232C.INIT performs as documented in the Lisp Library modules manual, except for the addition of the *LINETYPE* argument. *LINETYPE* should be SYNC or ASYNC (also accepted are the alternate spellings SYNCH and ASYNCH).

In order to use the RS232C port for communicating with a Pup Gateway it must be configured (using RS232C.INIT) in SYNC mode. In addition, the variable

RS232C-NETWORK [Variable]

should be set to T (which is the default setting on loading the RS232CNetwork module.

The variable

RS232C-NETWORK-AUTODIAL [Variable]

controls whether the system will attempt to start the network connection automatically when Lisp returns from a LOGOUT. If *RS232C-NETWORK-AUTODIAL* is set to NIL, you can cause the system to attempt to establish the connection manually by calling the function RESTART.ETHER.

The modem should be configured to dial on detecting an off to on transition of DTR. On a Codex 2260 modem this is DTR mode 108.1. Since the dialing and answering process can take a non-trivial amount of time, the variable

RS232C-NETWORK-DIALING-TIMEOUT [Variable]

indicates how many seconds the code will wait for the modem to signal that the connection has been established (by raising DSR) before timing out and continuing without the network connection. The default value is 30 seconds.

EXAMPLE

The very first time the code has been loaded the initialization sequence, when using Codex 2260 modems which can communicate at 9600 baud, would be

```
(RS232C.INIT 9600 8 'NONE 1 NIL 'SYNC)

(SETQ *RS232C-NETWORK-AUTODIAL* T)

(RESTART.ETHER)
```

Subsequently, whenever the Daybreak boots into Lisp, the network code will attempt to establish a phone connection.

To shutdown the phone connection you can call the function TURN.OFF.ETHER. The connection will also be broken when you log out of Lisp (assuming that the modem is configured to disconnect on loss of the DTR signal, which the Codex 2260 modems will do)

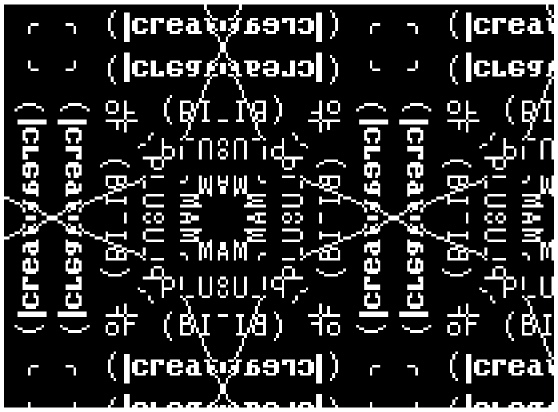
SCREENPAPER

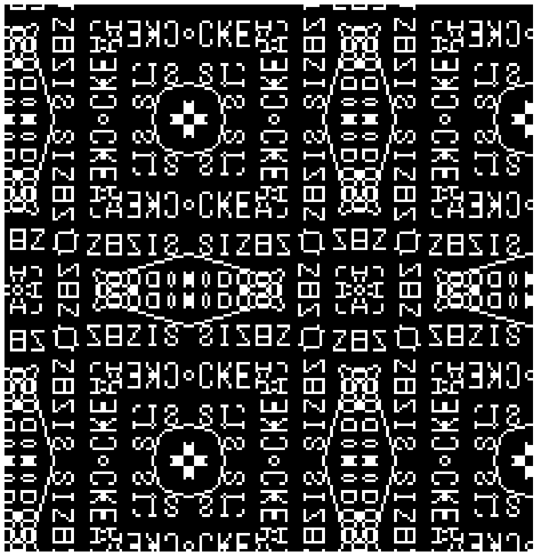
By: Larry Masinter (Masinter.pa@Xerox.com)

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).





SEARCHMENU

By: John Maxwell (Maxwell.pa)

Uses: DICTTOOL, DICTCLIENT, ANALYZER

INTERNAL

This document last edited on October 16, 1987

INTRODUCTION

The SearchMenu package implements a user interface to the relevance search capabilities of the dictionary server. Relevance search is a technique for finding items by giving examples of what you are looking for and having the search algorithm look for "similar" items. These items are then displayed in a FreeMenu which allows you to select those items which are actually relevant. You can then iterate on the new set of examples until no new relevant items show up.

The Dictionary Server currently supports three databases that can be searched using this technique: the WordNerd (based on the American Heritage Dictionary), the EtymologyNerd (based on the etymological portion of the American Heritage Dictionary), and the IRMNerd (based on the Interlisp Reference Manual). For example, if you were looking for for different types of gases, setting the database to the WordNerd and giving it example gases such as hydrogen and helium would produce oxygen, xenon, krypton, argon, neon, radon, flourine, chlorine, and nitrogen. Or, if you were looking for Interlisp functions that tested the equality of something, setting the database to the IRMNerd and giving example procedures like EQ and EQUAL would produce EQP, IEQP, FEQP, STREQUAL, EQMEMB and EQUALALL.

HOW TO USE THE SEARCH MENU

When you load the SearchMenu package, a Search Menu will appear in the lower left corner of the screen. (The Search Menu expands to display the results of a search, so you might leave it there until you see how big it can get.) The Search Menu has a number of commands on the top line, followed by a place to type in examples.

To try your first search, click the "Examples:" field and type "lion tiger". Then click the MATCH WORDS! command just above it. A little menu of databases will appear, asking you which database you want to search in. Select "DictServer: WordNerd". A message will then appear in the prompt window saying: "Searching in DictServer: WordNerd for words like: (tiger lion)." After about 30 seconds a new Search Menu will be created with a list of items that the Dictionary Server thinks is similar to a lion and a tiger. At the top of the list lion and tiger will already be selected.

To continue the search, scan the list of items for things that are relevant to you. If you are not sure what a word means, click the "def" button to its left, and the definition will be printed out in a separate window. If you want an item to be included in the next search, simply click at it and it will become inverted, like lion and tiger. If you click at an item twice, a line will be drawn through it to indicate that any keywords that it uses should have their weights reduced. Words that are neither highlighted nor struck out are ignored. When you are through examining the items, invoke the MATCH WORDS! command. You can iterate like this as often as you like.

Clearing The Search Menu

Before you start a new search, you should invoke the "CLEAR!" command to clear the Search Menu. The Search Menu caches some information about the search you are conducting which may interfere with your next search. Therefore, to be sure that you get a clean search, you should clear the menu.

Changing Databases

When you want to search for items in a new database, all that you need to do is click the "SET DATABASE!" command. This will cause a menu of databases to appear. Clicking one of the items in the menu will cause the Search Menu to search in that database from then on. Clicking outside the menu leaves the Search Menu in its current state.

The Key Menu

To the left of the commands in the first line is a command labelled "KEY MENU!". Invoking this command will turn the Search Menu into a key menu. The key menu will have all of the keys that were used in the last search, along with their weights. The weights are editable. If you want to see what would happen if you searched using different weights, simply edit the weights and then invoke the "MATCH KEYS!" command. Weights that are set to 0 or have no value are ignored.

If you want to see all of the uses of a particular key click the "uses" button to the left of that key. To get back to the example menu, invoke the "SAMPLE MENU!" command in the upper left of the Search Menu.

Logging a Search

If you wish to keep a log of each search, set the SearchMenu.LogData variable to T. From then on, the Search Menu will write the results of its search into a private stream. When you click the "CLEAR!" command, a TEdit window will be opened on the log and the log cleared in preparation for the next search.

THE PROGRAMMER’S INTERFACE

The Dictionary Server can be accessed directly through the following procedures:

(DICTCLIENT.MATCHWORDS *POSWORDS NEGWORDS MINWORD MAXWORD DICTIONARY*)[Function]

Takes a list of words to match, plus a list of words to ignore, plus the range of words that you are interested in, plus the database. The range of words lets you look at a search one chunk at a time: the first fifty (MINWORD = 1, MAXWORD = 50), then the next fifty (MINWORD = 51, MAXWORD = 100), and so on. The DICTIONARY is the name of the database to be searched in. Currently there are three possibilities: ’WordNerd, ’EtymologyNerd, and ’IRMNerd. If no data base is specified, the default is ’WordNerd.

(DICTCLIENT.WEIGHTEDSEARCH *WEIGHTEDKEYS MINWORD MAXWORD DICTIONARY*)[Function]

Takes a list of weighted keys to match, plus the range of words that you are interested in, plus the database. The weighted keys should be of the form ’((key1 weight1)(key2 weight2)...). The range of words lets you look at a search one chunk at a time: the first fifty (MINWORD = 1, MAXWORD = 50), then the next fifty (MINWORD = 51, MAXWORD = 100), and so on. The DICTIONARY is the name of the database to be searched in. Currently there are three possibilities: ’WordNerd, ’EtymologyNerd, and ’IRMNerd. If no data base is specified, the default is ’WordNerd.

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.

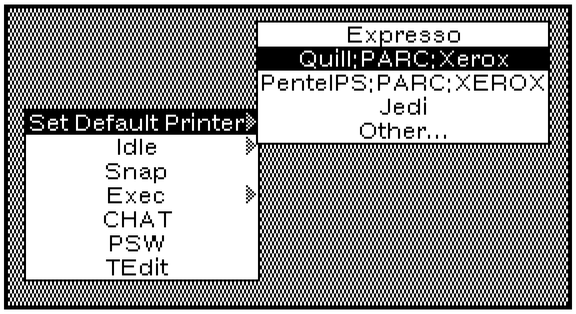
SETDEFAULTPRINTER

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") (PENTELPS: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 specifiy and 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 environemts 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]

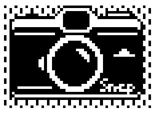
SNAPW-ICON

By: Randy Gobbel (Gobbel.pa)

Uses: nothing but basic window functionality

This document last edited on September 8, 1988.

INTRODUCTION

SNAPW-ICON creates an icon for shrunken screen snap windows. It looks like this: 

Everything else about screen snaps is as before.

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 ia 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.

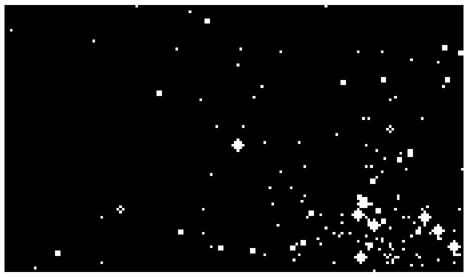
STARBG

By: Gregg Foster (Foster.PA@Xerox.COM)

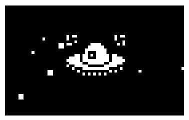
Upgraded for Medley by Larry Masinter (Masinter.PA@Xerox.COM)

STARBG creates a random star field for your screen background and 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.

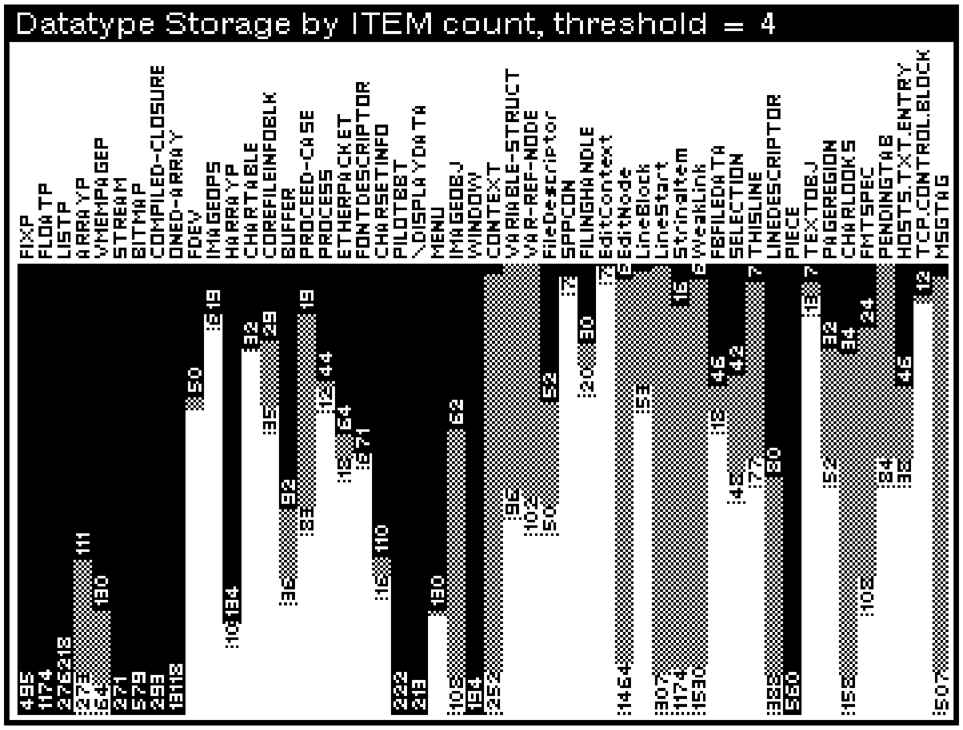
STORAGE

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.

STYLESHEET

By: Tayloe Stansbury

Unsupported

INTRODUCTION

Stylesheets are collections of menus. These collections pop up all at once in a group. This group does not disappear until all menus in it have been dealt with, and the user signals that he is done.

Stylesheets are intended to be used in situations wherein the computer wants an answer to several related questions all at once. One example is font selection. To select a font, the user needs to specify font family (Classic, Modern, etc.), font size (8 point, 10 point, etc.), and font style (bold, italic, etc.). Rather than prompt for each of these parameters in succession, one could use a stylesheet to prompt for it all at once.

When the stylesheet pops up, it will shade (preselect) default selections (if provided) in each of the menus. The user can either decide that the default selections are OK, or change them to suit his taste. (The default selection mechanism can be used to convey the current state of something the user is trying to change with the stylesheet: for example, the current looks of the text with which the user is dissatisfied.)

When the user is finished, he hits the DONE button and the stylesheet disappears, and the final selections are returned. There is also a RESET button. This is useful if the user has mucked up his selections and would like to reinstate the default selections. Finally, there is an ABORT button that if selected returns NIL from STYLESHEET and is intended to provide the user with a convenient way of aborting the selection. Note: This means that NIL can be returned from a call to STYLESHEET.

Menus in a stylesheet can be set up to accept exactly one selection (like a normal menu), less than two selections, or any number of selections. Menus that need not be filled in (i.e., can accept zero selections) have an attached CLEAR button, which can be used to remove selections made in that menu. Menus that can have more than one selection have an attached ALL button, which can be used to select all the items in the menu.

HOW TO MAKE A STYLESHEET

To create a stylesheet, call

(CREATE.STYLE *Prop1 Value1 Prop2 Value2 ... PropN ValueN*) [Function]

CREATE.STYLE accepts an arbitrary number of property-value pairs. Properties currently recognised are

ITEMS [Style Property]

A list of menus. Most menu format parameters contained in menu records are honored by the stylesheet package. WHENSELECTEDFNs are, of course, ignored.

SELECTIONS [Style Property]

A list of menu items, each one corresponding to a menu in ITEMS. The specified selections will be shaded in the appropriate menu, and will be the default selections. If not specified or too short, it will be filled out with NILs (no selection).

NEED.NOT.FILL.IN [Style Property]

A list of T or NIL or MULTI, each one corresponding to a menu in ITEMS. T indicates that the corresponding menu need not be filled in and will be given a CLEAR button. MULTI indicates that the corresponding menu can have any number of selections and will be given both a CLEAR button and an ALL button. If the list is too short, it will be filled out with NILs. If a single T or NIL or MULTI is given instead of a list, it will be replaced by a list of Ts or NILs or MULTIs, respectively.

TITLE [Style Property]

The title that will be given to the stylesheet. If no title is specified, the stylesheet will not have a title bar.

ITEM.TITLES [Style Property]

A list of strings or atoms to serve as titles over the menus. Items without titles specified will not have titles.

ITEM.TITLE.FONT [Style Property]

A fontdescriptor or other font specification which determines the font item titles will be printed in. If NIL, titles will be printed in DEFAULTFONT.

POSITION [Style Property]

The screen position (of type POSITION) of the lower left-hand corner of the stylesheet. If position is not specified, the function STYLESHEET will prompt for the postion (using GETBOXPOSITION). STYLESHEET will modify positions as necessary to ensure that the entire stylesheet will be on the screen.

Stylesheets can be modified by calling

(STYLE.PROP *Stylesheet Prop Newvalue*) [Function]

STYLE.PROP always returns the old value of the specified property of the specified stylesheet. If Newvalue is provided (even if NIL), it replaces the old value. If not provided, the old value remains. (Just like WINDOWPROP.)

To use the stylesheet thus created, call

(STYLESHEET *Stylesheet*) [Function]

This returns a list of selections the user made from the stylesheet. (If a selection is returned as NIL, that indicates that no selection was made.)

One can determine in advance of displaying a stylesheet how big it will be. (This may help in determining a reasonable screen position for the stylesheet.) The relevant functions are

(STYLESHEET.IMAGEWIDTH *Stylesheet*) [Function]

and

(STYLESHEET.IMAGEHEIGHT *Stylesheet*) [Function]

They return the width and height, respectively, of the stylesheet in pixels.

AN EXAMPLE

The package is located in STYLESHEET and STYLESHEET.DCOM. To familiarize yourself with its workings, you might want to load it and try the following example:

```
(SETQ FONT.STYLE
  (CREATE.STYLE
    'TITLE "Please select a font:"
    'ITEM.TITLES '(Family Size Face)
    'ITEM.TITLE.FONT '(Modern 12)
    'ITEMS
      (LIST
        (CREATE MENU ITEMS ← '(Classic Modern Terminal))
        (CREATE MENU ITEMS ← '(8 9 10 11 12 14))
        (CREATE MENU ITEMS ← '(Regular Bold Italic BoldItalic)))
    'SELECTIONS '(Modern 11 Regular)
    'NEED.NOT.FILL.IN 'T]

(STYLESHEET FONT.STYLE]
```

SuperParentheses

By: Andrew J. Cameron, III (Cameron.pa@Xerox.com or cameron@cs.wisc.edu)

Most useful when used with: WHO-LINE (LispUsers)

This document last edited on Oct 19, 1987.

INTRODUCTION

This file, when loaded, creates a readtable (named "LISP[]") for use with CommonLisp which contain SuperParentheses, that is, the left square bracket (LEFTBRACKET syntax class) and right square bracket (RIGHTBRACKET syntax class) available in InterLisp. CommonLisp does not give these two characters their "usual" definitions, so as to allow users to easily give these character any macro/syntax definition they might desire.

This readtable will appear on, and can be selected via, the "Rdtbl" menu provided by the WHO-LINE LispUsers utility.

One can also access this new readtable with:

```
        (IL:FIND-READTABLE "LISP[]")
```

These facilities obviate the need to store the readtable on a variable, as was done in an earlier version of this module.

INTERNALS

• The reading and writing of files using this readtable has not been tested or explored.

• SEdit is not too friendly to SuperParentheses.

• For more information, see Section 25.8.2 in both the IRM (InterLisp Reference Manual) and the Lyric Release Notes, and Section 22.1 in Steele (CommonLisp - The Language).

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. Retuns 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*. Retuns *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*. Retuns *difstats*.

(CLOCKTICKS *interval timerunits*) [Function]

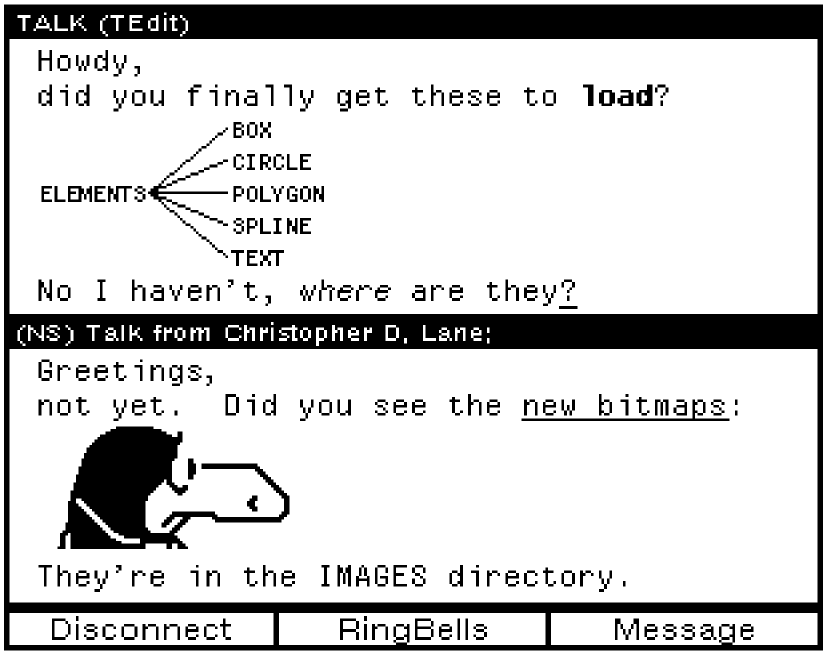
Returns the (machine dependent!) number of internal clock ticks over the interval. For instance, on the D'Lion, (CLOCKTICKS 2.5 'MINUTES) = 5211900.

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**?

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.

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:

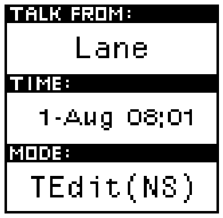
Host not found!	It could not find host address for the host or user name specified.
Can’t connect to host!	The remote workstation does not have the appropriate server loaded and/or running or does not have TALK loaded.
No answer from TALK service!	A connection was made, but no one responded (intentionally 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:



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

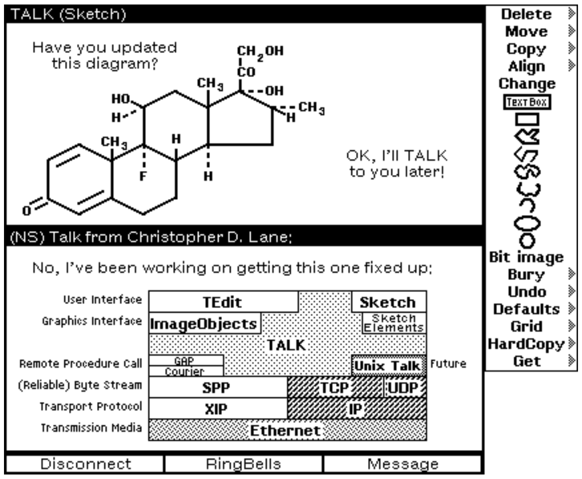
This service allows you to use the full capabilities of TEdit in your conversation, including: correcting mistakes anywhere in the document, changing character and paragraph looks, inserting ImageObjects, etc. Along with keyboard input, mouse selections and the caret are also visible to the remote user. 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 (Sketch)

Have you updated this diagram?

OK, I'll TALK to you later!

(NS) Talk from Christopher D. Lane:

No, I've been working on getting this one fixed up:

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

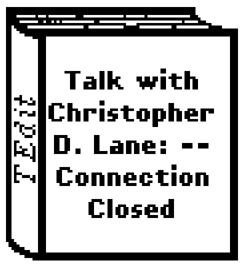
- The TTY service cannot backspace beyond the left margin (unlike other implementations).

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.

TEdit-Close-On-Shrink

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

Uses: TEdit

This document last edited on August 4, 1987.

INTRODUCTION

TEdit has the unfortunate habit of keeping the file you are editing open when you shrink the TEdit window. For users of the FileCache this has the unfortunate sideeffect of preventing a newly saved file from being written out. In places where many people are editing files on file servers can lead to an excessive number of open files. TEdit-Close-On-Shrink attempts to handle some of the problem by persuading TEdit to close unmodified files when the edit window is shrunken.

USE

Load the module. It will install itself in the appropriate places (for the curious: that's `TEDIT.CREATEW`, `\TEDIT.CREATEW.FROM.REGION`, and `\TEDIT.REOPEN.STREAM`; and the shrink function `\TEDIT-CLOSE-ON-SHRINK`).

TEdit Dorado Keys

A set of convenience keys for the Dorado and TEdit.

INTERNAL

This package defines a number of meta-keystrokes as TEdit commands, providing much of the functionality of the 1186's expanded keyboard. An effort was made to keep the command set compatible with SEdit's for similar functions.

In a number of cases, the meta-lower-case and meta-upper-case commands are different--typically, the lower-case command turns some attribute (like boldness) on, and the upper-case command turns it off.

Here's the set of defined functions:

meta-u	
meta-U	UNDO
meta-f	
meta-F	FIND
metaa	
meta-A	ABORT (i.e., do a GET)
meta-s	
meta-S	SUBSTITUTE
ESC	REDO (note that this changes the default definition, which is EXPAND)
meta-x	
meta-X	EXPAND (also on control-X)
meta-n	
meta-N	Move to NEXT fill-in blank
meta-c	
meta-C	Change margins, rotating thru centered, left, right, just.
meta-b	Turn BOLD on
meta-B	Turn BOLD off
meta-i	Turn ITALIC on
meta-I	Turn ITALIC off
meta-=	Turn strike-thru on
meta-+	Turn strike-thru off
meta--	Turn underline on
meta-_	Turn underline off
meta-^	Superscript (or de-subscript)
meta-|	Subscript (or de-superscript)
meta-space	Return to default font/weight/slope/etc.
meta-?	Show current font in prompt window.
meta-(	Insert parentheses around the current selection
meta-"	Insert neutral double quotes (ASCII ") around the current selection
meta-'	Insert real double quotes (" and ") around the current selection

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 " is selected in "the set X is empty" then pressing the SUPERSCRIPT button produces "the set 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 exdents 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
#]	exdents 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:
OPEN	^o (or ^O)
SAME	META
FIND	finds item in TEdit menu
AGAIN	ESC
DELETE	DEL
COPY	SHIFT
MOVE	CTRL-SHIFT
PROP'S	META or LOCK depending on switches

NEXT	#n (or #N)
EXPAND	^x (or ^X)
HELP	#?
MARGINS	#[(unnest (which is shift-MARGINS on the Dandelion) is #])
FONT	top blank
KEYBOARD	middle blank
UNDO	bottom blank
STOP	^G
shift-**STOP**	#^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 exdent 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.

TEdit-Line-Numbering

By: Randy Trigg (Trigg.pa)

This document last edited on May 9, 1988

INTRODUCTION

TEDIT-LINE-NUMBERING enables the automatic conversion of a TEdit selection into multiple lines of specified width each ending in a carriage return and prefixed by a line number. For example, TEDIT-LINE-NUMBERING converted the following piece of transcript:

C: We have to be able to check that, within the memory I, I claim.

(1.0)

M: Check whether Tore is a graduate student? I think we can do that (.) I mean

C: Yea I know but more we have to be able to, within the memory somehow, recognize that (.) because of this constraint being in the memory, we have to check that the time matches Wednesday morning. We have to add this constraint to the time and see if (.) the time is not overconstrained.

----

into:

```
021   C:    We have to be able to check that, within the memory I, I claim.

022   (1.0)

023   M:    Check whether Tore is a graduate student? I think we can do that
024         (.) I mean

025   C:    Yea I know but more we have to be able to, within the memory
026         somehow, recognize that (.) because of this constraint being in
027         the memory, we have to check that the time matches Wednesday
028         morning.  We have to add this constraint to the time and see if (.)
029         the time is not overconstrained.
```

INTERFACE

(MAKE-LINE-BREAKS *TextStream WidthInInches Device StartLineNum LineNumDigits FirstLineParaLooks OtherLineParaLooks InsertExtraTabFlg InsertExtraCRFlg*)

TextStream should be a TEXTSTREAM of an open TEdit. Line breaks and line numbers will be inserted for the text in the current selection. WidthInInches is the number of inches wide the resulting lines will be when printed to Device. Device should be one of the litatoms DISPLAY, INTERPRESS, or PRESS. StartLineNum will be the line number used for the first line formed. LineNumDigits is the number of digits used to print the line numbers. FirstLineParaLooks should be a standard TEdit paralooks proplist for the first line formed from every paragraph. OtherLineParaLooks will be used for the remaining lines of each paragraph. InsertExtraTabFlg, if non-nil, will cause an extra tab to be inserted after the line number. InsertExtraCRFlg, if non-nil, causes an extra carriage return to be inserted between paragraphs. (If this last Flg is on, then FirstLineParaLooks and OtherLineParaLooks are probably equal.)

For example, the conversion depicted above was done with the following call (where SS is bound to an open textstream):

(MAKE-LINE-BREAKS SS 5 'INTERPRESS 21 3 '(LINELEADING 2 PARALEADING 5) '(LINELEADING 2 PARALEADING 0) T)

The algorithm used is (pretty much) as follows: First, place a temporary marker at the end of the selection. For each paragraph in the selection, insert a line number LineNumDigits wide followed by a tab. Move one character at a time through the paragraph adding up STRINGWIDTHs until reaching WidthInInches (converted to appropriate units for Device). Then move back to last whitespace and insert a carriage return. Change the paralooks of the paragraph just formed according to FirstLineParaLooks. Insert the next line number followed by one or two tabs (depending on value of InsertExtraTabFlg) and continue adding up STRINGWIDTHs. When WidthInInches is reached, insert a carriage return and change the paralooks to OtherLineParaLooks. Continue in this manner till the end of the paragraph. If InsertExtraCRFlg is non-nil, then insert an extra carriage return. Continue with the rest of the paragraphs until encountering the marker at the end of the selection. Delete the marker and quit.

NOTES

It's a good idea to have a couple of tabs set for the selected text, though MAKE-LINE-BREAKS will use the default tab setting if you don't.

It tries to do proper measurement of embedded tabs, but this hasn't been extensively tested.

BUGS

IT'S INCREDIBLY SLOW! This is because we use only calls available from the TEdit programmers interface. Things could be significantly sped up by walking the piece table like TEdit does when printing, but it's alot of work to figure out how to do that and anyway I'd prefer that this tool only call advertised functions.

Doesn't handle imageobjs.

TEDIT-PF-SEE

By:

Ron Kaplan

This document created in January 2022.

This tiny package adds alternatives to the PF and SEE commands that produce their output in scrollable read-only TEDIT windows rather than the unscrollable EXEC window. The new commands are tf (for t(edit)f(unction) and ts for t(edit)s(ee)

```
tf   FUNCTION  (FILELIST)  (REPRINT)                                    [command]
```

prints the definitions of `FUNCTION` that appear in the files in `FILELIST`, with a separate `TEDIT` allocated for each definition. If `FILELST` is not provided, then `WHEREIS` is used to locate the defintions, just as with `PF`. By default the definition characters are simply copied from the source file to the `TEDIT` stream, but if `REPRINT` is T the definition is read and then reprinted. This produces useful output for definitions that were not pretty-printed. Also, if `FUNCTION` is not provided, definitions for the last invocation will be reprinted.

```
ts  FILE (WINDOW)
```

shows the contents of `FILE` in a scrollable read-only `TEDIT WINDOW. This uses the function TEDIT-SEE` (also used for the `FILEBROWSER See` command), which interprets any font changes if `FILE` is a Lisp source file.

`TEDIT-PF-SEE` loads the `REGIONMANAGER` package, and the default regions for tf and ts are of type `PF-TEDIT` and `SEE-TEDIT` respectively. The function SET-TYPED-REGIONS can be used to predefine the regions where the output for tf and ts will appear.

TEdit-Process-Killer

By: Steve Bagley (Bagley.pa) and Randy Trigg (Trigg.pa)

This document last edited on Apr 23, 1987

INTRODUCTION

TEDIT-PROCESS-KILLER provides a simple interface to removing and restoring the process of a TEdit window. The processes of TEdit windows can be killed selectively or a TEDIT-KILLER process can be started to keep the total number of active TEdit processes at or near some threshold level. TEdit processes are automatically rebuilt when you button in their windows.

INTERFACE

(KILL-PROCESS-OF-TEDIT-WINDOW *WINDOW*) [Function]

kills the processes associated with the main window of WINDOW, and all of the attached windows. Each process is killed in such a way that the TEdit can be restarted. It is not an error to call this function on a TEdit whose process has already been killed.

(WITHOUT-TEDIT-PROCESS *WINDOW*) [Function]

returns T if this window does not have a process, because the process was killed by KILL-PROCESS-OF-TEDIT-WINDOW, NIL otherwise.

(RESTART-PROCESS-OF-TEDIT-WINDOW *WINDOW*) [Function]

restarts the TEdit processes for the main window of WINDOW and all attached windows if the processes have been killed by KILL-PROCESS-OF-TEDIT-WINDOW.

(START-TEDIT-KILLER) [Function]

starts up a process called TEDIT-KILLER which wakes up at regular intervals to kill off the least recently used TEdit processes. There are two global vars available to the user to affect its operation:

TEDIT-PROCESS-LIMIT [Variable]

Defaults to 10. The preferred threshold of running TEdit processes. Every time TEDIT-KILLER wakes up, it kills off enough TEdit processes to bring the total down to this limit.

TEDIT-KILLER-WAIT-TIME [Variable]

Defaults to 10000. The time in milliseconds between wake-ups of TEDIT-KILLER.

(STOP-TEDIT-KILLER) [Function]

kills any running TEDIT-KILLER process.

NOTES

In order to force a TEdit to be killed off when shrunk, simply do

```
(WINDOWADDPROP <Win> ’SHRINKFN (FUNCTION KILL-PROCESS-OF-TEDIT-WINDOW))
```

and, if you like,

```
(WINDOWADDPROP <Win> ’EXPANDFN (FUNCTION RESTART-PROCESS-OF-TEDIT-WINDOW))
```

BUGS

We don’t kill lafite sendmessage processes.

TEKTRONIX

By: Jim Blum (Unsupported)

Last Modified: James Turner (Turner:Lexington:Xerox)

TEKTRONIX 4010 EMULATOR

FILES: TEK4010CHAT, TEK4010

This document last edited on: 9-May-88 23:52:18

There are two LispUsers Modules, TEK4010 which takes an INSTREAM and OUTSTREAM as arguments and emulates a TEKTRONIX 4010 storage tube terminal, and a version called TEK4010CHAT which works with CHAT. The details on how and what a TEKTRONIX 4010 terminal does are described in a TEKTRONIX 4010 users manual. The CHAT program is described in the Interlisp Reference Manual. This document will point out the differences from the default DM2500 emulated terminal which CHAT uses, and the relevant issues unique to running the TEK4010 emulator under Interlisp.

TEK4010 is called as follows:

(TEK4010 *INSTREAM OUTSTREAM*) [Function]

where OUTSTREAM must be a displaystream (or window). This version does not fully support the TEKTRONIX 4010 terminal. Specically, the graphic input mode (which displays the crosshair and waits for a key to be typed), and does not support the double column text mode. This version supports a limited scaling feature. Both the TEKTRONIX 4010 X and Y coordinates are divided by one global integer called TEKPTSPERPOINT which can be set(q) by the user. No attempt is made to scale the text or line spacing (leading) in this version, nor does reshaping the window automatically change TEKPTSPERPOINT.

The CHAT version (TEK4010CHAT) unlike the DM2500 emulator, runs in non-scroll, paint mode, with the right margin set at the width of the window. Although not elegant, this is how the TEK4010 terminal works and there may be cases where users are dependent on its inherent mode of operation. The visible implications of this on INTERLISP are: The screen is not cleared or scrolled when a linefeed is received on the last line. Graphics and text are overlayed (OR'd) using the PAINT mode in BITBLT, since selective areas on a storage tube cannot be erased. Graphics are scaled to the window size and so is the position of the second margin, but the characters in the font (GACHA 10) are not scaled. Scaling is changed when the CHAT window is reshaped. In order to get accurate positioning of both the text and graphics, the Interlisp window size should match the TEK4010 screen resolution which is 1024 horizontally by 768 vertically.

In order to run the TEK4010 emulator under CHAT, the following recipe must be followed:

1. Make sure you have the latest version of CHAT loaded.

2. Load the TEK4010 emulator module, ie, (LOAD ’TEK4010CHAT.LCOM)

3. Edit the variable CHAT.DISPLAYTYPES (ED ’CHAT.DISPLAYTYPES) to add one or more lists of the form (host number TEK4010) where host is the name of the host that you want to chat to with this emulator. Use the number 4010 for the number field which would be used by the CHAT.SETDISPLAYTYPE function. Add as many entries as there are hosts you want to use the TEK4010 emulator with.

4. Then bug CHAT in the background MENU and select or enter the host that you want to chat to.

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.

TKDORADO

By: Mike Dixon (MikeDixon.pa)

INTERNAL

TKDorado: TEditKey for the Dorado keyboard

TKDorado makes the full range of TEditKey commands available from the Dorado keyboard. To do so it adds TEditKey bindings for a number of meta-control key combinations:

`Meta-Control-`**V**:	set to default (vanilla) looks
`Meta-Control-`**B**:	bold on
`Meta-Control-`**N**:	bold off
`Meta-Control-`**I**:	italics on
`Meta-Control-`**O**:	italics off
`Meta-Control-`**D**:	overbar on
`Meta-Control-`**F**:	overbar off
`Meta-Control-`**G**:	strikethru on
`Meta-Control-`**H**:	strikethru off
`Meta-Control-`**J**:	underlining on
`Meta-Control-`**K**:	underlining off
`Meta-Control-`**[**:	smaller font
`Meta-Control-`**]**:	larger font
`Meta-Control-`↑:	superscript
`Meta-Control-`_:	subscript
`Meta-Control-`**C**:	center/justify/left this paragraph

The functions which change character looks operate on the current selection, if any characters are selected, and otherwise change the caret looks.

The new key bindings are also added to the help menu.

In addition, TKDorado rebinds the ESC key to be REDO, rather than Expand (which is still available as Control-X), and closes the DLion keys window (if TEditKey has left it lying around).

TMAX

Chapter 1. Introduction

TMAX stands for Tedit Macros And eXtensions and it enhances TEdit by providing a convenient way to do things such as numbering, indexing, creating a table of contents, and more. At SUMEX these sort of operations are done with Scribe[1]. Scribe is a powerful document preparation language but it consumes all together too many cycles on our mainframe. Furthermore with Scribe you must hardcopy your document to see what it looks like. If you want to change the format, you must add the appropriate commands to the Scribe input file, run Scribe again, and hardcopy the output file. This sort of "batch processing" wastes both yours and the mainframe's time. TEdit is a WYSIWYG (What You See Is What You Get) text editing and formatting system. You see what your document will look like while you are creating it.

TMAX makes no attempt to mimic Scribe in TEdit nor does it have any facility to translate a Scribe .MSS (i.e. source) file into a TEdit file. Rather it implements some of the more commonly used features of Scribe in TEdit. Currently there are four main areas in TMAX; indices, numbering, endnotes, and forward and backward references. The TMAX features described here plus the editing and formatting features already available in TEdit make it an attractive alternative to Scribe. For more information on TEdit, please read the first part of the **Text Editing** section in **The Lisp Library Packages Manual**.

All the features described here are merely additions to a TEdit document. TMAX does not alter your text in any way. When you invoke one of these features, TMAX inserts a "special character" (i.e. an Image Object[2]) into your document at the current cursor position. These "special characters" may appear to be strings but they are really just single characters. This means you can delete any TMAX feature you add by simply deleting the corresponding "special character" just as you would delete any other character. The features described in this document are also used throughout this document. Rather than including pictures of each menu (and there are lots of them), we have decided to show you how to pop-up these menus yourself. This document is both an explanation and example of how to use TMAX. We suggest you read this document under TMAX/TEdit and try these features as you read about them. This document was written assuming you are reading it in a TEdit window.

Chapter 2. Menus

When you first load TMAX.DCOM, a new item called **TMAX Menu** is added to the main TEdit pop-up menu. Since you probably used this menu to **Get** this document, you may have already seen this new item. Buttoning **TMAX Menu** brings up the TMAX menu which is attached to the top of your TEdit window. Now you can invoke the TMAX features by simply buttoning items in this menu. You can remove this window by right buttoning the mouse in its title bar and selecting Close (just as you would remove any other TEdit menu). The small window just above this window is called the "prompt window". TMAX uses this window both to prompt you for text input and to report current values and status. Please pop up the TMAX Menu now by pointing the mouse at the black bar above this window, left or middle buttoning it, and selecting **TMAX Menu**.

There are three types of fields in the TMAX menu; labels, values, and commands. You select items by pointing the mouse at the item and pressing either the left or middle button.

1) The fields on the far left of each line (Miscellany:, References:, etc.) are simply labels that specify the nature of the items on the rest of that line. You cannot select these label fields.
2) There are eight items that are used to set values. Five of the items are toggle switches, two define output filenames, and one pops up a menu of possible settings.
   2.a) The five toggle switches are ***Set AutoUpdate***, ***NGroup Menu***, ***Text Before***, ***Text After***, and ***Manual Index***. They are all in a bold italic font to distinguish them from other items. These 2-way toggle switches appear in normal video (as they are

above) when they are off and in reverse video when they are on. Buttoning these items complements their current setting.

2.b) **TOC Filename:** and **Index Filename:** are used to define the name of the Table-Of-Contents and Index file respectively. When these items are buttoned, the cursor appears to the right of the colon and TMAX waits for you to specify a filename. On a **Get** these items are defaulted to the name of the input file with the extensions .TOC and .INDEX.

2.c) The last value item is **Reference By** and the light face field following it is its current setting. Buttoning this item pops up a menu of possible settings.

3) All the other items (everything except labels and values) are commands. Buttoning these items invokes the corresponding TMAX feature. These items turn to reverse video while they are active and then return to normal video when they finish.

Many of the TMAX features use pop-up menus. Buttoning the mouse outside any pop-up menu is equivalent to no selection and will make that menu disappear. Whenever TMAX prompts you for a new value, it always displays the current value in the prompt window.

We will now discuss each item in the TMAX menu in detail. Since many of the items interact with other items, it is difficult to find a starting point. Instead we have described the TMAX menu row by row and left to right within each row. We suggest you first peruse this document to get a general idea of the features and then focus on the areas that interest you.

Chapter 3. Miscellany

3.1 Update

When you use TMAX to do some sort of numbering, TMAX inserts a "marker" rather than the actual number. It does this for speed since it is much faster to insert a marker rather than figure out the corresponding numeric value each time. Buttoning **Update** will convert all these markers to their corresponding (and consecutive) numeric values. In addition, any References (see chapter 4) to these numbers will be updated. Finally, if the Endnotes (see chapter 5) have already been inserted, **Update** will reinsert them but this time with the actual Endnote numbers. Currently the only numbering constructs in TMAX are Endnotes, Number Groups, and References to them. Please button **Update** now and watch all the changes.

3.2 Undo Update

Buttoning this item undoes everything that **Update** does. That is it converts all the Endnote and Number Group numbers back to their corresponding markers along with any References to these numbers. If the Endnotes have already been inserted, **Undo Update** will reinsert them but this time with the Endnote numbers replaced by their markers.

3.3 Set AutoUpdate

When this toggle switch is on, all numbering markers are immediately updated to their corresponding numeric value as they are inserted. Note that it only updates inserted markers; it does not automatically update any markers in a TEdit file that you load via **Get**. We do not encourage setting this switch for large documents. This switch causes TMAX to check every number it generated in the entire document whenever a new number is inserted. For large documents this could take some time.

3.4 Current Date/Time

Buttoning this item inserts the current date into your document. For example, the date enclosed in the following parentheses (April 22, 1987) was inserted by buttoning **Current Date/Time**. Middle buttoning this inserted date pops up a Date/Time menu. If you'd like to see this menu, go ahead and middle button the date in the parentheses. This menu allows you to change the format of the date/time, replace the date with the time, or update an old date/time to the current date/time. The font is the same as the font currently in effect in your TEdit document when the date/time was inserted.

Chapter 4. References

These commands allow you to reference Endnotes and Number Groups by either their numeric value or by the page number they appear on. A Reference is simply an association between a "Tag" and a reference to that Tag. To assign a Tag, middle button the Endnote or Number Group marker (or its numeric value if it has already been updated). A menu will pop up and one of the items will allow you to define a Tag for the Endnote or Number Group. If the Endnote or Number Group is already tagged, the menu will allow you to change or delete the Tag. All the Tags in a document must be unique and TMAX will not allow you to create a Tag that is already defined. If you COPY a tagged Endnote or Number Group within the same window, TMAX will remove the Tag on the copied object. If you COPY a tagged object to another TEdit window, TMAX will preserve the tag assuming that Tag name isn't defined in the other TEdit window. The font of a Reference is the same as the font currently in effect in your document regardless of what font is used to display the Endnote or Number Group.

As an example, we have tagged both the Endnotes chapter (a Number Group on page <5/Page>) with "EN Chap" and the Endnote at the end of this sentence with "EN Note"[3]. Now we can reference the Endnotes chapter as 5 and the previous Endnote as 3. Also the page number of the Endnotes chapter above was generated by a Reference to "EN Chap" by page instead of by value. You will have to button **Update** to convert the "<Chapter/Value>" and "<Note#/Value>" to their respective numeric values. Of course, if ***Set AutoUpdate*** was on, the "<Chapter/Value>" and "<Note#/Value>" would be converted as soon as they were inserted.

4.1 Reference

TMAX will prompt you for a Tag name when you button this item. A bare carriage return cancels this command. It doesn't matter if the Tag you specify is defined yet or not. If the Tag is not defined, the Reference marker is "<Reference Tag/Type>" where Tag is the Tag name and "Type" is either Value or Page. In this case the Tag name is embedded in the marker. If the Tag is defined, the Reference marker is either <Note#/Type> for Endnotes or the Number Group marker enclosed in angle brackets (e.g. <Chapter/Type>). In this format there is no indication of what the Tag name is. If you middle button a Reference marker (whether it is updated or not), TMAX will display the corresponding Tag in the prompt window. All Reference markers that have not been updated are enclosed in angle brackets (i.e. < and >) to distinguish them for normal text.

4.2 Previous References

Buttoning this item brings up a menu of all the defined Tag names in alphabetical order. You can then create a Reference to one of these Tags by simply buttoning the corresponding Tag in this menu. Although it doesn't matter if you define the Tags or References to these Tags first, we suggest you define the Tags first. Once the Tags are defined you can use this command to easily create the References to these Tags. If you button **Previous References** now you will see all the Reference Tags defined in this document.

4.3 Reference By

This item allows you to select whether you want References by numeric value (of the Endnote or Number Group) or by the page number (on which the Endnote or Number Group appear). Buttoning this item pops up a menu with three items; Ask, Value, and Page. If you select Value or Page then all References you add will be of the type you specified. If you select Ask then TMAX will prompt you for the type every time you make a Reference. As you can see in the TMAX menu above, the default Reference type is by Value.

There is one minor inconvenience with doing forward references by page. You must do a dummy hardcopy and then button **Update** before the real hardcopy. The reason is TEdit only knows the current page number while it is hardcopying a document. As TMAX encounters Endnotes and Number Groups, it saves their corresponding page numbers. If you reference an Endnote or Number Group that TMAX hasn't seen yet then it can't possibly know the corresponding page number. In this case, TMAX will change the Reference marker to the Endnote's or Number Group's numeric value followed by "/Page" both enclosed in angle brackets (e.g. <1.2/Page>) to indicate that the page number is not known yet. This only applies to forward references by page. You don't have to do a dummy hardcopy if you are referencing by value or doing backward references by page.

Chapter 5. Endnotes

These commands allow you to generate a numbered list of notes. Endnotes are like footnotes except the numbers and corresponding text appear at the end of the document rather than the bottom of the page. (TMAX does not support footnotes yet.) The Endnote numbers are consecutive and always start at 1. If you add an Endnote in the middle of a document, all the following Endnote numbers will be adjusted automatically. Suppose for example you have three Endnotes and you add another between Endnotes 1 and 2. The new Endnote becomes 2 and the Endnotes that were formerly

2 and 3 are now 3 and 4. This will happen when you button **Update** (or immediately if ***Set AutoUpdate*** is on). The font of the Endnote numbers, text, and title line is determined by the **Set Style** command and the default font is Gacha 10 Standard.

If you middle button an inserted Endnote marker, a menu will pop up allowing you to define a Tag for this Endnote marker or edit the text associated with this Endnote marker. If this Endnote already has a Tag, this menu will give you the option of changing, deleting, or displaying the Tag or editing the Endnote text. If you choose to edit the Endnote text, you will be prompted to open another TEdit window where the text will be displayed. When you are done editing the text, move the mouse to the title bar (i.e. the thick black bar at the top of the new TEdit window) and left or middle button the mouse. A menu will pop-up giving you the option of saving or aborting the changes. There is a sample Endnote at the end of this sentence you can play with[4].

5.1 Endnote

Buttoning this item inserts an Endnote marker at the current cursor position. The Endnote marker is a superscript "Note#". You will also be prompted for the text associated with this Endnote.

5.2 Insert Endnotes

This item inserts the title line "Notes" followed by the Endnote numbers and their corresponding text after the last line in your TEdit document. If you have already inserted the Endnotes and button this command again, TMAX will delete the old Endnotes and then reinsert them. It does this in case you have added or deleted any Endnotes since the last time they were inserted.

5.3 Delete Endnotes

Buttoning this item undoes what **Insert Endnotes** does. It deletes the title line "Notes" and the Endnote numbers and text that follow from the end of your document.

If you already inserted the Endnotes and wish to add more text at the end of your document, you ***must*** first delete the Endnotes with this command. If you don’t, the additional text at the end of your document will be deleted when the Endnotes are reinserted. Suffice it to say this is due to the way Endnotes are implemented rather than a bug in TMAX. A future version of TMAX will fix this awkwardness.

5.4 Set Style

Buttoning this item pops up the Endnote Fonts menu. This menu has three items; Number, Title, and Text. You can use this pop-up menu to change the font of the Endnote numbers, title line, and text. The title line is always "Notes". When you select one of the items from this menu, the current font of that item will be displayed in the prompt window before TMAX prompts you for the new font.

Chapter 6. Number Groups

The Number Group (NGroup) commands allow you to number any arbitrary objects in your TEdit document. You define the hierarchy, font, and format of each NGroup member and then insert them wherever you want something numbered. The NGroup numbers are consecutive and start at 1 by default although you can change the starting value. This allows you to number individual pieces of a document without having to load the entire document. Some NGroup values depend on other NGroup values. For example, the sections of this document each begin with the corresponding chapter number. Each time the chapter number changes, the section number is reset to its starting value. You can nest NGroups as deep as you like. If you insert a new NGroup member in the middle of a document, all the following NGroup members that depend on this new member will be adjusted when you button **Update** (or immediately if ***Set AutoUpdate*** is on). This makes it trivial to number (and renumber) things. All you do is insert an NGroup member wherever you want something numbered and TMAX takes care of all the rest. The default NGroup font is Gacha 10 Standard.

6.1 NGroup Menu

When this toggle switch is on, the NGroup menu "graph" is displayed in a window attached to the top of the TMAX menu window. The Number Group graph is a tree structure showing the hierarchy and order of all the Number Group members. If you haven’t buttoned ***NGroup Menu*** yet, please do so now.

6.2 New NGroup

This item allows you to create new Number Group members. When it is buttoned, TMAX first turns on ***NGroup Menu*** if it was off so you can see the NGroup graph. It then prompts you for the name of the NGroup member. Finally TMAX prompts you for the parent of this new NGroup member by popping up a menu of the known NGroup names and you select the parent from this menu. By parent we mean the node to the immediate left in the NGroup graph. If you don't select any parent (i.e. button the mouse outside the pop-up menu) the new NGroup becomes a "top level" node. A top level node is one whose parent is the boxed **NGroups** (at the far left). The NGroup graph is always built from left to right. Since the first NGroup member defined will have no parents, TMAX just adds this member to the graph. You can create as many NGroup members as you like even if you don't use them all. As long as there is at least NGroup in you document, the entire NGroup graph will be saved/restored over a **Put**/**Get**.

6.3 Text Before

When this toggle switch on, you will be prompted for a preceding text string each time you insert a NGroup member. This text string becomes part of the inserted NGroup marker. You can use this string as a heading for the chapter, section, figure, example, etc. you have numbered. The text string is always printed in the same font as the corresponding NGroup member and TMAX always inserts one space between the preceding text string and the NGroup marker. If you would like more spaces, you must put them in the ***Text Before*** string yourself. Any tabs in this string are automatically converted to spaces. The default ***Text Before*** string is the name of the NGroup. In this document all the Chapter NGroup markers were inserted with ***Text Before*** on and the Section NGroup markers were inserted with ***Text Before*** off.

6.4 Text After

When this toggle switch on, you will be prompted for a succeeding text string each time you insert a NGroup member. This text string becomes part of the inserted NGroup marker. You can use this string as a heading for the chapter, section, figure, example, etc. you have numbered. The text string is always printed in the same font as the corresponding NGroup member and TMAX always inserts one space between the NGroup marker and the succeeding text string. If you would like more spaces, you must put them in the ***Text After*** string yourself. Any tabs in this string are automatically converted to spaces. There is no default ***Text After*** string. In this document all the Chapter and Section NGroup markers were inserted with ***Text After*** on.

6.5 Changing the default Font and Format

Consider the Number Group graph in this document. The boxed **NGroups** at the far left is a special node and buttoning it does nothing. The order of the nodes in a branch is important but the order of the branches themselves isn't. If you would like to see a more complex NGroup graph then just add some more members to this graph. Don't worry; adding extra NGroup members doesn't affect anything.

When you create NGroup members by buttoning **New NGroup**, you are actually defining "prototype" NGroups. The font and format of these prototypes determine the font and format of the NGroups you insert into your document. To change the font and/or format of a prototype NGroup member, simply point the mouse to the member name in the Number Group menu graph and ***middle*** button it. A menu will pop up with four items; Change Font, Show Font, Change Format, and Show Format. Buttoning Show Font or Show Format will display the font or format of the selected prototype NGroup in the prompt window.

Buttoning Change Font or Change Format allows you to change the font or format of the selected prototype NGroup. This is a *global* change. If you change the font/format of a prototype NGroup, TMAX will apply that change to every occurrence of that NGroup in your document. Before TMAX prompts you for the new font/format, it always shows you the current font/format in the prompt window. Change Font and Change Format both have subitems that allow you to change part of the font/format without changing anything else.

The subitems for Change Font are Family, Size, and Face. If you button one of these subitems, TMAX will pop up the corresponding menu and only that part of the font will be changed. If you button Change Font instead, TMAX will prompt you for all three values via pop-up menus.

There are seven subitems for Change Format. You can change any part of the format by selecting one of these subitems. If you button Change Format instead, TMAX will prompt you for all seven values. Six of the values use pop-up menus and the other requests input in the prompt window. The seven parts of the prototype format are:

i: Delimiter Before
This is the delimiter that precedes the NGroup. TMAX pops up a menu of commonly used delimiters. You can select one of these or select Other in which case TMAX will prompt you for a delimiter string in the prompt window. The default Delimiter Before is a null string.

ii: Display Type
This is how the NGroup's numeric value is displayed. TMAX pops up a menu of the various ways a NGroup can be displayed. These ways are as an Arabic numeral, a null string, or an upper/lowercase letter or Roman numeral. The default is as an Arabic numeral.

iii: Delimiter After
This is the delimiter that follows the NGroup. It uses the same mechanism as the Delimiter Before. The default Delimiter After is a period (i.e. ".").

iv: Abbreviate Level
Normally a NGroup's value is the concatenation of all its parents values plus its own value. This subitem allows you to specify how far up the NGroup branch to go when computing a NGroup's value. TMAX pops up a menu containing this NGroup name and all its parents. You control how much of the NGroup value to display by selecting the first NGroup to be used in this NGroup's value. Note that this subitem doesn't actually change the NGroup value; it only determines how much of this value to display. Since top level NGroups only have a single value, you cannot abbreviate them. The subitems listed here are numbered with the NGroup "Sect Cntr" which as been abbreviated such that it only prints its own value without any of its parents values.

v: Starting Value
This is the starting value of the NGroup. TMAX prompts for the new value in the prompt window rather than using a menu. The default Starting Value is 1.

vi: Table-Of-Contents
This is a flag that says whether or not this NGroup will be included in the Table-Of-Contents file should you decide to create one. The default is to include all NGroups in the Table-Of-Contents file.

vii: Manual Index
This is a flag that says whether or not this NGroup should be included in the manual-style index. Note that this item is only offered if the ***Manual Index*** (see section 8.4) toggle switch is on. The default is to not include any NGroups in the manual-style index.

6.6 NGroup Delimiters

The Delimiter Before/After construct allows TMAX to support any numbering format. Since the default Delimiter Before is a null string and the default Delimiter After is a period, the default number format is of the form "1.", "1.2.", "1.2.3." etc. You may have noticed that this document uses this default form for the Chapter numbers but not for the Section numbers. To see how we did this just middle button Chapter in the NGroup graph above and select the Show Format item from the pop up menu. Then do the same for Section. The field following "Display=" consists of the Delimiter Before, the Display Type, and the Delimiter After in that order.

There is one small anomaly with delimiters, namely the case where one NGroup has a Delimiter After (e.g. Chapter) and the following NGroup has a Delimiter Before (e.g. Section). If the Delimiter Before is not a null string then it will *always* override the Delimiter After in the preceding NGroup regardless of what the Delimiter After is. For example, if the Delimiter After for Chapter was a colon and the Delimiter Before for Section was a dash then chapter numbers would look like "1:" and section numbers would look like "1-2.".

6.7 Inserting Number Groups

You create NGroup members by buttoning **New NGroup**. To insert an NGroup member into your document, simply point the mouse to the appropriate name in the Number Group menu graph and ***left*** button it. The NGroup member name enclosed in square brackets (e.g. [Chapter]) will be inserted at the current cursor position in whatever font you have specified for this member. All NGroup markers that have not been updated are enclosed in square brackets to distinguish them from normal text. TMAX will warn you if you insert NGroups out of order. For example, in this document you should not insert "Section" until you insert "Chapter" because the value of Section depends on the value of Chapter.

6.8 Customizing inserted Number Groups

Although the font/format of the prototype NGroup determines the font/format of the inserted NGroup, you can change the font and certain parts of the format after the NGroup is inserted. Any change you make is *local* to that particular NGroup; it does not affect any other NGroups. If you middle button an inserted NGroup, a menu will pop up allowing you to define a Tag and show/change the font and format of the selected NGroup. If the NGroup already has a Tag, this menu will give you the option of changing, deleting, or displaying the Tag. The Show Font and Change Font items work exactly the same as those for the the prototype NGroups but the Show Format and Change Format are slightly different. The Starting Value and the Table-of-Contents and Manual Index flags can only be set for the prototype NGroup. You cannot change these for a particular inserted NGroup.

There are six subitems for Change Format. You can change any part of the format on an inserted NGroup by selecting one of these subitems. If you button Change Format instead, TMAX will prompt you for all six values. Four of the values use pop-up menus and the other two request input in the prompt window. The six parts of an inserted NGroup's format are:

i: Delimiter Before
This works exactly the same as the Delimiter Before subitem for the prototype NGroups except the change doesn't affect any other NGroups.

ii: Display Type
This works exactly the same as the Display Type subitem for the prototype NGroups except the change doesn't affect any other NGroups.

iii: Delimiter After
This works exactly the same as the Delimiter After subitem for the prototype NGroups except the change doesn't affect any other NGroups.

iv: Abbreviate Level
This works exactly the same as the Abbreviate Level subitem for the prototype NGroups except the change doesn't affect any other NGroups.

v: Text Before
This allows you to add, change, or delete the text string preceding the NGroup regardless of the setting of the ***Text Before*** toggle switch. TMAX will prompt you for the new string in the prompt window. A bare carriage return deletes the text before the NGroup. If you do specify a string, remember that TMAX will always append a space to the end of the string. Unlike inserting a NGroup with ***Text Before*** on, there is no default string.

vi: Text After
This allows you to add, change, or delete the text string following the NGroup regardless of the setting of the ***Text After*** toggle switch. TMAX will prompt you for the new string in the prompt window. A bare carriage return deletes the text after the NGroup. If you do specify a string, remember that TMAX will always append a space to the beginning of the string.

6.9 Pruning the NGroup graph

As mentioned before, the entire NGroup graph is saved/restored over a **Put**/**Get** even if some of the NGroup members aren't used in the document. We have described how to add members to this graph but not how to delete them. TMAX has two ways to delete unused NGroup members from the graph.

The first way is to Copy text (including the inserted NGroups) from one TEdit window to another. TMAX will only copy enough of the NGroup graph to handle the NGroups in the copied text. For example, you can remove every unused NGroup member from the graph by copying the entire document to another TEdit window.

The other way is to set the global variable TMAX.PRUNE.NGRAPH to T, open a TEdit window, and then **Get** the document. On a **Put** TMAX writes out the NGroup graph data structure. On a **Get**, TMAX checks this flag and if it is true, it creates the NGroup graph from the NGroups that appear in the document rather than from the NGroup graph data structure. There is a potential problem with using this feature. TMAX creates the NGroup graph from the first occurrence of each NGroup in the document. If the first occurrence of a particular NGroup has been modified (see section 6.8) then TMAX will use this modified format as the new prototype format. There are no problems if the first occurrence of each NGroup has not been modified.

Chapter 7. Contents File

One of the benefits of using NGroups is the Table-Of-Contents (TOC). By default all NGroups are included in the TOC along with their corresponding ***Text Before*** and/or ***Text After*** strings if any. If you don't want certain NGroup members included in the TOC, you can specify this by changing the NGroup's prototype format. Although this document has several NGroups, we have decided to include only the Chapter and Section NGroups in the TOC. The page numbers in the TOC are always printed in the Gacha 10 Standard font.

7.1 Create TOC

Buttoning this item creates a TEdit Table-Of-Contents file. You must first specify the name of the TOC file via the **TOC Filename:** item. Each line in the TOC consists of the ***Text Before*** string (if any), the NGroup number, the ***Text After*** string (in any), a dotted leader, and the page number on which this NGroup appears. Note that the TOC file itself does not contain any TMAX features. It is a simple TEdit text file.

7.2 View TOC

Buttoning this item first creates the TOC file (via **Create TOC**) and then prompts you for a TEdit window where TMAX displays the TOC file it just created.

7.3 TOC Filename:

This item allows you to specify the name of the TOC file. When this item is buttoned, the cursor will appear just to the right of the colon. You then type the name of the TOC file and terminate it with a carriage return. You can edit this filename string anytime. If you have already terminated the string, just button the item again. The editing features available here are the same as those available in the EXEC window. If you **Get** a TEdit/TMAX file, the **TOC Filename:** defaults to that file with a **.TOC** extension.

NOTE... You must hardcopy your document before creating the TOC file. The reason is TMAX needs the page numbers for the TOC file but TEdit only knows the page numbers while the document is being hardcopied. During the hardcopy process, TMAX saves the page number for each NGroup member. If you create the TOC without first hardcopying your TEdit document, the page numbers in the TOC will be NIL.

Chapter 8. Indices

TMAX allows you to insert index requests into your document and create a sorted file of these indices including the page number each index appears on. There are two types of index requests; simple and extended. The format of the index marker depends on the type of index request. But, regardless of the type of index, it is important to note that the index markers are only displayed in the TEdit window. If you hardcopy the TEdit document, you will ***not*** see these index markers. TMAX always encloses index requests in curly braces (i.e. { and }) to distinguish them from normal text. **Update** and ***Set AutoUpdate*** have no effect on index requests.

8.1 Index

When this item is buttoned TMAX will prompt you for the index "Key". It will then insert the marker "{Index key}" into your document at the current cursor position. TMAX uses these keys to sort the indices and the sorting is case independent. The key is also printed in the Index file along with the page number(s) it appears on. Currently all simple indices are printed in the Gacha 10 Standard font. For example, we have indexed the phrase "Indexing requests" in the following parentheses (). Of course you will not see this index marker if you hardcopied this document. You can change the index key by middle buttoning the index marker. TMAX will bring up a one item menu (Change Index). If you button this item, TMAX will prompt you for the new index key.

8.2 Extended Index

This is a fancy form of indexing. When this item is buttoned TMAX first prompts you for the key to sort on. TMAX then prompts you for the "Entry" and its font. This is what is printed in the Index file instead of the key. If you do not specify an entry, it defaults to whatever the key is and the font defaults to Gacha 10 Standard. Finally, TMAX prompts you for the index page number option. There are three options; print the normal page number, print a fixed page number that you supply, or don't print any page number at all. TMAX then inserts the index marker "{Index Key=key, Entry=entry, Option}" into your document. The option is "Yes" if the page number is to be included in the index file, "No" if the page number is not included, or the numeric value if a fixed number is to be used in the index file. For example, we have (extended) indexed the word "Spies" but we want "Boris & Natasha" printed instead in the Helvetica 12 Italic font. The extended index is enclosed in the following parentheses (). Of course you will not see this extended index marker if you hardcopied this document. You can change any of the fields in an extended index request by middle buttoning the index marker. TMAX will bring up a one item menu (Change Extended Index). If you button this item, TMAX will prompt you for the new index key, entry, font, and number values.

8.3 Known Indices

Buttoning this item brings up a menu of all the indices and extended indices specified so far in alphabetical order. You can insert another **Index** or **Extended Index** request by simply buttoning the appropriate item in this pop-up menu. This makes it trivial to index the same items throughout a document. Indices are simple items but extended indices have subitems because several extended indices can have same key but completely different entries, fonts, and/or page number options. To insert an extended index, you must button the appropriate subitem; buttoning the extended index item has no effect. The extended index subitem shows the entry, font, and page number option. If you button **Known Indices** above, you will see the two indices used here as examples.

8.4 Manual Index

When this toggle switch is on, the page numbers in the index file are printed in "manual format". By manual format we mean something like "III:25.7" for chapter 3, section 25, page 7 (assuming the chapter's format has been changed to print Roman numerals following by a colon). You specify which NGroup members are included in the manual index page number. To include a NGroup member in the manual format page number, first make sure this toggle switch is on. Then change the format of the NGroup members you want included. The code to change the NGroup format checks the ***Manual Index*** switch setting. If the switch is on, the code asks if you want the selected NGroup member included in the manual index. You can only include the members of one major branch of the NGroup graph in the manual index page numbers. Using this document's NGroup graph as an example, you can include either or both members in the Chapter-Section branch ***or*** the single member in the NG Format branch but ***not*** both. TMAX will only allow you to include an NGroup member in the manual index if there are no other members included yet or the included members are in the same branch as the new member. The reason is there is no correlation between the numbers in disjoint branches in the NGroup graph. If you don't specify manual indexing, TMAX defaults to "book format" indexing. With book format the index file page references are just the page numbers themselves.

Chapter 9. Indices File

These commands allow you to write out an index file sorted by the index keys. As we mentioned before, the sorting is case independent. The page numbers (for both "book" and "manual" style) are always printed in the Gacha 10 Standard font.

9.1 Create Index

Buttoning this item creates a TEdit sorted index file. You must first specify the name of the index file via the **Index Filename:** item. For simple indices each line in the index file consists of the index key followed by the page number(s) on which this index key appears. Extended indices are treated a little differently. Each extended index is printed on a separate line. Note that the index file does not contain any TMAX features. It is a simple TEdit text file.

9.2 View Index

Buttoning this item first creates the index file (via **Create Index**) and then prompts you for a TEdit window where TMAX displays the index file it just created.

9.3 Index Filename:

This item allows you to specify the name of the index file. When this item is buttoned, the cursor will appear just to the right of this item. You then type the name of the index file and terminate it with a carriage return. You can edit this filename string anytime. If you have already terminated the string, just button the item again. The editing features available here are the same as those available in the EXEC window. If you **Get** a TEdit/TMAX file, the **Index Filename:** defaults to that file with a **.INDEX** extension.

NOTE... You must hardcopy your document before creating the index file. The reason is TMAX needs the page numbers for the index file but TEdit only knows the page numbers while the document is being hardcopied. During the hardcopy process, TMAX saves the page number(s) on which each index appears. If you create an index file without first hardcopying your TEdit document, the page numbers in this index file will be NIL.

Chapter 10. Specifying a Font

TMAX uses the same mechanism whenever it prompts you for a font. To specify a font you must select a value from each of three different menus. The first pop-up menu is used to specify the font family. The font families are Classic, Gacha, Helvetica, Modern, and Times Roman. After you select the family, the font size menu pops up. This menu contains the sizes 6, 8, 10, 12, 14, 18, 24, and 36. After you select the size, the font face menu pops up. The font faces are Standard, Italic, Bold, and Bold Italic.

The default for any value not specified (i.e. the mouse is buttoned outside the pop-up menu) is whatever it was before. The default font always starts out as Gacha 10 Standard. Therefore, if you select a new family (say Helvetica) and button the mouse outside the font size and face menus, the new font would be Helvetica 10 Standard. Now if you were to button the mouse outside the font family and size menus and select a new face (say Bold), then the new font would be Helvetica 10 Bold.

Some TMAX functions (like **Set Style**) require you to select values from three different menus. Other functions (like changing a NGroup font) allow you to change one of the three fields directly without changing the other two. This is exactly equivalent to selecting a value from one menu and buttoning the mouse outside the other two menus.

Chapter 11. Random thoughts and hints

o You can Copy any of the TMAX features. from one TEdit window to another and TMAX will automatically set up the internal data structures necessary to support that feature in the destination window. For example, if you Copy a NGroup marker, TMAX will automatically set up the NGroup graph. Note that it only sets up enough to support what was copied; TMAX does not set up the entire NGroup graph.

o TMAX does not support Move. If you want to move any TMAX features, you will have to Copy and then delete the features.

o If you just want to select a TMAX feature, you should do so with the left mouse button. The right button will select the feature as well but it also causes TMAX to pop up a menu. Every TMAX feature pops up a menu if it’s middle buttoned.

o Regarding forward page References and doing a "dummy" hardcopy, we do the dummy hardcopy to a file whose device is {NODIRCORE} and whose extension corresponds to the eventual hardcopy device (e.g. {NODIRCORE}X.INTERPRESS). NODIRCORE is the bit bucket and the extension causes TEdit to load the appropriate fonts so the real hardcopy will go a little faster.

o Remember that changing the font/format of a prototype NGroup will also change every occurrence of that NGroup in the document. Therefore you should settle on the font/format of the prototype NGroups before you start customizing any inserted NGroups (see section 6.8).

o Occassionally TEdit (not TMAX) has problems displaying some TMAX features. For example, if you change the prototype NGroup font then every occurrence of that NGroup also changes. Sometimes it looks like only the first occurrence has changed when in fact they all have. If you

Redisplay the window, it will look like it should. It is a good idea to Redisplay the TEdit window whenever you think it doesn’t look right.

TMENU

By: Mark Stefik, Daniel G. Bobrow, and Chris Tong (Stefik.pa@Xerox.com)

The TMENU package provides the following features:

TMenus.

These are interactive menus intended for reducing the amount of typing to an Interlisp-D program. When an item is selected in a menu, an expression is inserted into the TTY input buffer. It can be used to simplify the entry of common LISP commands or long names.

Windowshades.

Windowshades are a modification that can be made to any window in order to conserve screen space. They make a window "roll up" when not in use, leaving behind only the title. When the mouse is clicked inside the remaining bar, the window unrolls for interaction, and rolls up again at completion. Windowshades can be used with TMenus.

Msgs to the Prompt window.

Two functions are provided for clearing and printing to the Interlisp-D prompt window. These functions take an arbitrary number of arguments and preserve the black background shade of the window.

INTERACTION WITH TMENUS

TMenus are placed on the display under program control using the functions described below.Once a menu is on the screen, items may be selected using the LEFT mouse button. This causes some text to be inserted into the teletype buffer. In the default case, this text is just the item in the window, but it can alternatively be the result of evaluating an expression. In the default case, the text is followed by a blank space in the buffer, but it can be followed alternatively by an arbitrary string (such as the empty string or a carriage return). Non-default cases are controlled by arguments to the TMenu function.The MIDDLE mouse button is used for user-interactions that change the menu. When the middle button is depressed inside a menu, another pop-up menu (a meta-menu!) appears that provides several options for changing the menu, such as adding or deleting items. One of the options is to recompute the set of items by evaluating an expression associated with the menu.The RIGHT mouse button is used for the usual window commands. These commands work inthe standard way except that when a TMenu is reshaped, internal functions are invoked to adjust the configuration of rows and columns in order to create a menu that is visually appealing.

MAIN FUNCTIONS

TMenu (*itemExpr title displaySpec windowShadeFlg buttonFndefaultTrailerString*)

[Function]

TMenu is the function for creating a menu in a window on the display. Its arguments are as follows:

itemExpr an expression for computing the list of items that is saved with the menu. In the usual case,itemExpr is a list, whose items are either atoms or lists of the form:(displayThis evalThis comment trailerString)where displayThis is the form displayed in the menu, evalThis is the form evaluated to createthe text for the input buffer, comment is displayed in the prompt window if the LEFT button is held over an item for an extended period, and trailerString is the string inserted after anexpression in the input buffer. Most of these fields are optional. The default value for evalThisis the entry in displayThis. The default trailerString can be specified for a TMenu in thedefaultTrailerString argument above. Otherwise it is a space if the expression is an atom, and the empty string otherwise. This definition of fields for menu items is compatible with the usual set for the Interlisp-D menu package, with the addition of the trailerString field.

Special cases:

If itemExpr is an atom, it must be the name of a global variable to be evaluated to yield the list of the form described above (e.g., MYFNS). If itemExpr is a list and the first element of the list has a functional definition, then that function is evaluated to yield the list.This is intended for cases where the list is to be computed.

title The title of the menu. This title is used as the title of the window containing the menu.

displaySpec This argument has several possible interpretations that control the display of the menu. IfdisplaySpec is a region, then the window for the menu is placed in that region. If displaySpec is a number, that number is used as the number of columns in the menu display and a minimum size window is allocated for displaying the entire menu. The user is prompted with a ghost box to place the menu on the display. If displaySpec is T, then the number of columns is computed by TMenu assuming a maximum of 15 rows per column and the user is prompted for placement as before. If displaySpec is NIL, then the user is prompted to place a bounding box for the menu and TMenu tries to compute an arrangement of rows and columns that is visually pleasing.

windowShadeFlg If T, the window containing the menu is augmented with a window shade so that the menu"rolls up" if not in use. If windowshadeFlg is NIL, the menu is placed in a window on the screen.

buttonFn Optional argument that allows the caller to specify his own function for handling the LEFT andMIDDLE buttons.

defaultTrailerString Optional argument that allows the caller to specify the default string to follow each item printed. Can be overridden for specific items by the trailerString argument in the itemExpr.

EXAMPLE

The following expression:(TMenu ’MYFNS "Common Fns" T) would create a window titled "CommonFns" and display the list of functions in the window. The window would contain columns of up to 15 functions each, and would be placed on the screenunder user control. If the user later used the MIDDLE button to add or delete items from the menu, then the list in the variable MYFNS would be updated as the menu is updated.

MakeFileMenus (*fileName*) [Function]

MakeFileMenus is the function for creating a set of menus for the functions and variables in afile. A window is created for each menu. The menus are placed under user control and are all given window shades. The argument fileName is the name of a file.

Example

If the file is MYFILE, then the fns on MYFILEFNS and the vars on MYFILEVARS would be displayed in menus. MakeFileMenus would look for file commands of the form (FNS * FnsLst)and (VARS * VarsLst) on the command.

CloseFileMenus (*fileName*) [Function]

Closes all of the TMenu windows associated with the given file.

Window ShadesMakeWindowShade (*window*) [Function]

Modifies the given window to provide a window shade. If the window argument is NIL, then the window is selected which is under the cursor. If the window argument is T, it waits for the CTRL key to be depressed, and then selects the window under the cursor.Prompt Window FunctionsPROMPT (arg1 arg2 arg3 ...)Prints an arbitrary number of arguments to the prompt window, after first clearing the window.A call with no arguments simply clears the window.

CPROMPT (*arg1 arg2 arg3* ...) [Function]

Same as prompt except the arguments are printed centered in the window.

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.

Sender: SenderNameTooLong%SUMEX-AIM.STANFORD:EDU:Xerox
Date: 5 Jan 88 18:39:54 PST (Tuesday)
Subject: Interlisp to Commonlisp conversion package
From: darrelj%sm.unisys:COM
To: info-1100%sumex-aim.stanford
cc: darrelj%sm.unisys:COM

Return-Path: <@SUMEX-AIM.Stanford.EDU:sdcrdcf!sm.unisys.com!darrelj@PRC.Unisys.COM>
Redistributed: Info-1100-Redistribute^.X
Received: from SUMEX-AIM.Stanford.EDU by Xerox.COM ; 05 JAN 88 18:38:18 PST
Received: from burdvax.PRC.Unisys.COM (PRC-GW.PRC.UNISYS.COM) by SUMEX-AIM.Stanford.EDU with TCP; Tue, 5 Jan 88 17:44:10 PST
Received: by burdvax.PRC.Unisys.COM (burdvax) [5.54/1.0]   id AA15346; Tue, 5 Jan 88 20:44:28 EST
Received: by rdcf.sm.unisys.com (sdcrdcf) (5.51/Domain/jpb/2.7)   id AA28679; Tue, 5 Jan 88 16:35:37 PST
Message-Id: <8801060035.AA28679@rdcf.sm.unisys.com>
Received: from XAVIER by sdcrdcf with PUP; Tue, 5 Jan 88 16:35 PST
Original-Date: 5 Jan 88 16:34 PST (Tuesday)

Interlisp-D into Xerox Commonlisp, we developed a collection of tools
to automate the conversion as much as possible.
These have been placed in {parcvax.xerox.com}/lisp/exchange.
While we at Unisys have reasonable confidence in these tools, they are
being made available with no promises of accuracy, completeness or
support (though we would appreciate feedback).

The tools run in Xerox Lyric Common Lisp. The following files are parts of it:

TRANSOR -- A slightly modified version of Transor, to fix a few
Lyric-related problems and provide the ability to emit a
DEFINE-FILE-INFO expression and to use the value of TRANSOUTREADTABLE
as the output readtable. We used TRANSOR because we were familiar
with it, and it handles a lot of details needed to safely and surely
traverse the code to be translated. The biggest impediment to adding
to the transforms is that they are specified as teletype editor
commands, and only old-time Interlispers have much experience with those.
TSET -- The same version dating back to 1979. This is the part of
transor used for interactively developing and testing translation rules.
TRANSOR.LCOM -- contains the compilation of BOTH the above files.
TO-COMMONLISP.XFORMS -- translation rules for 428 functions, 98
remarks and 4 auxiliary functions. It covers are large portion of
Interlisp, including most Clisp constructs, and specifically handles
any function with the same name in both Interlisp and Commonlisp, so
that holes in a translation should result in calls to undefined
functions. In many cases, nice transformations are used for easy
cases, and ugly ones only for hard cases. This file sets
TRANSOUTREADTABLE to be a copy of the XCL readtable which is case
SENSITIVE, MYLOAD below reads it case INsensitive, so the resulting
file will ultimately lose most case distinctions on reloading into
Xerox Lisp (or other common lisps). This was a much debated point
internally, but this seemed the best of three bad possibilities (e.g.
print one of
Cased AS ORIGinal   which becomes CASED AS ORIGINAL on load,
|Cased| AS |ORIGinal|, or
CASED AS ORIGINAL

INTERLISP-COMMONLISP.TEDIT -- A document describing the
transformations and formacro.

LOADTRAN -- contains a few functions which prevent many breaks on
loading the translated file. The function MYLOAD is intended to load
a translated file.
LOADTRAN.DFASL -- compiled version

FORMACRO and FORMACRO.DFASL -- Still another portable iteration macro
for commonlisp. Its main claims are almost 100% compatibility with
the semantics of the Interlisp-Clisp FOR (especially when used the the
XFORMS which fix a few incompatibilities); and user extensibility
(unfortunately not compatible with IL:I.S.OPR). Embedded keywords
(e.g. IN, COLLECT) may be in any package.

COMMON-MAKE and COMMON-MAKE.LCOM -- still another version of code to
generate a more "common" source file. It handles more filepkg command
types than most. Also, when used with COMMENTHACKS will successfully
print ALL comments in semicolon format. Call
IL:COMMON-MAKEFILE(file). It checks the MAKEFILE-ENVIRONMENT property
to select a package and base.

COMMENTHACKS and COMMENTHACKS.LCOM -- patches to the prettyprinter and

to the DEFUN editor. The prettyprinter patches will print Interlisp (* --) comments as semicolon comments when *PRINT-SEMICOLON-COMMENTS* is 'IL:ALL.
This file also redefines the ED method for DEFUNs so that the initials and date of editing get updated for DEFUNs just as Interlisp has always done for FNS.

Because of the way things developed, these tools are not as fully integrated as they could have been. If we were doing it over, the TRANSOR step could have more carefully coordinated the new COMS so that COMMON-MAKE would be able to do the right thing. As it stands, the COMS generally have to be edited to change FNS to FUNCTIONS, etc, but you tend to need a few iterations of editing things before the compiler is completely happy anyway.

The steps needed to do translations are roughly as follows:

```
(LOAD 'TRANSOR.LCOM)
(LOAD 'TO-COMMONLISP.XFORMS)
(SETQ FIXSPELLDEFAULT 'N)  ;; Otherwise DWIM gets too clever
(SETQ XlatedRecords NIL)   ;; This is currently set to records
specific to the system we translated.
```

TRANSOR files containing record declarations. The records MUST be translated before any code containing create/fetch/replace since the translation depends on the type of records. Also, the record declarations should be LOADED. In a large translation effort, save a file containing all needed declarations and the value of XlatedRecords computed by translating them.

```
(TRANSOR 'file1) ...  ;; results in file1.TRAN and file1.LSTRAN, see
TRANSOR documentation.
```

To load translated files into a fresh xerox lisp system:

```
>From an XCL exec:
(IL:SETPROPLIST '*COMMENT* (IL:GETPROPLIST 'IL:*))
(IL:PUTASSOC '*COMMENT* 'IL:* IL:PRETTYEQUIVLST)
(LOAD 'LOADTRAN)
(SETQ IL:*DEFUALT-MAKEFILE-ENVIRONMENT* '(:READTABLE "XCL" :PACKAGE ???))
(SETQ IL:CMLRDTBL (IL:FIND-READTABLE "XCL"))
(LOAD 'FORMACRO.DFASL) if used interlisp for's
  ;; may need to import USER:FOR depending on packages you've set up.
(MYLOAD 'translated-records)
(MYLOAD 'file1.tran) ...
```

A little work with ED and FILES? and you should be able to save a commonlisp version of your files (well, OK, a lot of work).

Suggestions and questions to one of:
darrelj@RDCF.SM.UNISYS.COM or darrel@CAM.UNISYS.COM,
or fritzson@bigburd.prc.UNISYS.COM

----- End Forwarded Messages -----

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 cribsheet for the Turbo-Window KeyChords. Some addition capabilities not listed here are given on that cribsheet. 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 cribsheet by depressing the HELP key and right-buttoning on the background. Next, make a copy of the cribsheet by depressing SHIFT (the left shift key) and right-buttoning on the cribsheet. Drop the new copy of the cribsheet by releasing all keys and buttons and the pressing the left mouse button. [Note: The cribsheet 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 *x y viewport*) [Function]

(TWODGRAPHICS.MOVETOPT *position viewport*) [Function]

Here position is a POSITION in world coordinates

(TWODGRAPHICS.RELMOVETO *dx dy viewport*) [Function]

(TWODGRAPHICS.RELMOVETOPT *dposition viewport*) [Function]

(TWODGRAPHICS.DRAWTO *x y width operation viewport color dashing*) [Function]

(TWODGRAPHICS.DRAWTOPT *position width operation viewport color dashing*) [Function]

(TWODGRAPHICS.RELDRAWTO *dx dy width operation viewport color dashing*) [Function]

(TWODGRAPHICS.RELDRAWTOPT *dposition width operation viewport color dashing*) [Function]

(TWODGRAPHICS.DRAWLINE *x1 y1 x2 y2 width operation viewport color dashing*) [Function]

(TWODGRAPHICS.DRAWBETWEEN *position1 position2 width operation viewport color dashing*) [Function]

(TWODGRAPHICS.DSPRESET *viewport*) [Function]

Does a ‘‘DSPRESET‘‘ on the VIEWPORT

(TWODGRAPHICS.DSPFILL *region texture operation viewport*) [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 *position glyph viewport operation*) [Function]

Bitblts glyph to position with operation, with glyph centered at position.

Several functions provide access to the world to stream transformations.

(WORLDTOSTREAM *position viewport oldposition*) [Function]

Position is in world coordinates. Oldposition is smashed if provided.

Returns the corresponding position in stream coordinates.

(WORLDREGIONTOSTREAMREGION *region viewport*) [Function]

Region is in world coordinates

Returns the corresponding region in stream coordinates

(WORLDTOSTREAMX *x viewport*) [Macro]

Returns x in stream coordinates.

Uses unboxed floating point arithmetic

(WORLDTOSTREAMY *y viewport*) [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 inprovements 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: Allthe 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.

UNDIGESTIFY

By: Steven C. Bagley (Bagley.pa)

This document last edited on May 27, 1986

INTRODUCTION

This Lafite package allows you to unpack an Arpa Network digest, such as AIList, into its constituent messages. An new item, "Undigest," is placed on the browser menu. When a single message (presumably a digest) is selected, clicking on this item will delete the selected message, and append the constituent messages to the end of the mail folder. If the selected message is not a digest, or is a digest in a format that cannot be parsed properly, then a message will be printed and nothing will happen to the mail folder.

USER OPTIONS

`*DELETE-DIGEST-FLAG*`, if `T` means that the digest message should be deleted if it is successfully parsed. The default is `T`.

`*MOVE-TO-FIRST-DIGEST-MESSAGE-FLAG*`, if `T` means to select the first constituent message, if `NIL` means to select the first undeleleted message after the digest message. The default is `NIL`.

`*DONT-UPDATE-HEADERS-FLAG*`, if `T` means not to copy the To: field from the digest to each constituent message. The default is `NIL`.

PLANNED ENHANCEMENTS

Inserting the contained messages immediately after the digest, rather than appending.

Moving the contained messages to a different mail folder.

NOTES

Many digests are not in the correct format. The parser used in this program tries to be very forgiving, and hence, is relatively slow (about 10 seconds to parse a digest on a Dorado). If everyone adhered to RFC934, the parser could be optimized for speed, but, alas, this is not the case.

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 compatable with the UNIX™ facility of the same name. For details of the file format see the UNIX™ manual page on ‘uuencode’.

(UUENCODE *FILES INTO-FILE*) [Function]

Encodes the files named by FILES into INTO-FILE. FILES may be either a list or files or a single file name. Note that UNIX™ uudecode 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™ compatability 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™ uudecode 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

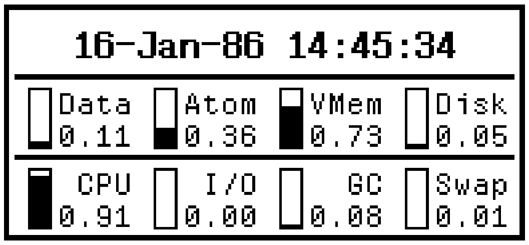
By: Johannes A. G. M. Koomen
(Koomen.wbst@Xerox or Koomen@CS.Rochester)

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:



```
16-Jan-86 14:45:34
Data   Atom   VMem   Disk
0.11   0.36   0.73   0.05
 CPU    I/O     GC   Swap
0.91   0.00   0.08   0.01
```

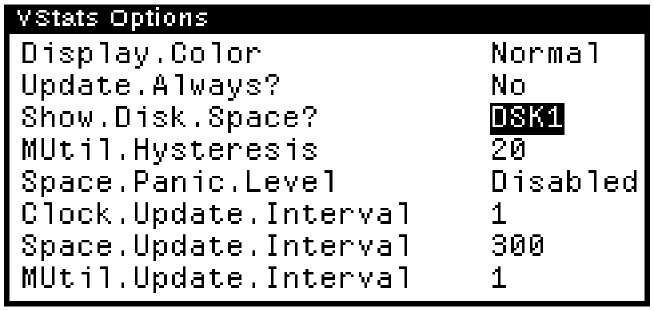
DESCRIPTION

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?         DSK1
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).

WALKFILES

Johannes A. G. M. Koomen
(Koomen.wbst@xerox or Koomen@cs.rochester.edu)

October 27, 1989

SUMMARY

WALKFILES is a facility for searching loaded files for arbitrary objects. It complements MasterScope, in that it can do some things that MasterScope can't (like looking for arbitrary substrings in functions); other things Masterscope can do as well or better. WALKFILES does not require an analysis of the files prior to action.

DESCRIPTION

(WALKFILES *pattern file{s} editcommands confirmflg quietflg filepackagetypes*) [Function]

Invokes WALKDEFS on all the objects on each *file{s}* of the types given in *filepackagetypes*. If *file{s}* is NIL, the value fo FILELST is used. If *filepackagetypes* is NIL, all the filepackage types in PRETTYTYPELST are used.

(WALKDEFS *pattern name{s} filepkgtype editcommands confirmflg quietflg*) [Function]

Walks over the *filepkgtype* definition of each name in *name{s}* looking for *pattern*. Pattern can be anything acceptable to EDITFINDP. For each occurrence, first prints the name (unless *quietflg* = *T*) and then prints the occurrence if *editcommands* is NIL, or invokes the editor in interactive mode if *editcommands* is T (asking for confirmation if *confirmflg* = T) , or applies *editcommands* to the name and *filepkgtype* if *editcommands* is a function, or otherwise invokes the editor with *editcommands*.

EXAMPLES

(WALKFILES 'elseif 'MYFILE T) [Example]

Brings up the editor on every definition in MYFILE which contains the symbol elseif.

(WALKFILES 'FOO) [Example]

Prints every occurence of any symbols with FOO as a substring in every definition in any file in FILELST.

```
(LET (THEFNS)                                                                    [Example]
    (WALKFILES  '(*ANY*  $FOO$  BAR)  NIL '(LAMBDA (FN) (PUSH THEFNS FN))  NIL  T  'FNS)
    THEFNS)
```

Collects all functions containing either the symbol BAR or any symbols with FOO as a substring on every file in FILELST.

WDWHACKS

By: Atty Mullins (Mullins.pa@Xerox.com)
Revised by: Ron Kaplan (kaplan@parc.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**, **SEDIT**, and **FILEBROWSERS**, 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 one of the **INSPECTORS**, **SEDIT**, or **FILEBROWSERS** subitems is selected, all open (not shrunken) windows of the indicated type 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 typesacript window).

WHEELSCROLL

By Ron Kaplan

This document was last edited in June 2023.

This small file adds the ability to scroll (scrollable) windows by rotating the wheel on a wheel mouse or by moving (2?) fingers on a track pad. The capability is enabled when `WHEELSCROLL.LCOM` is loaded. It is toggled on and off by

`(ENABLEWHEELSCROLL ON)` [Function]

Initially `(ENABLEWHEELSCROLL T)`.

The vertical scrolling speed is controlled by the variable

`WHEELSCROLLDELTA` [Variable]

The number of points to scroll for each click of the wheel. Higher values give faster scrolling. A negative value reverses the scrolling direction. Initially value: 20.

`HWHEELSCROLLDELTA` [Variable]

If non-NIL, this is the delta used for horizontal scrolling.

Implementation:

Lisp receives a key transition on `PAD1` or `PAD2` for vertical scrolling when the wheel rotates and no other keys are down. `(ENABLEWHEELSCROLL T)` modifies the keyaction table so that it maps these transitions to characters 156 and 157. Those characters are defined as interrupts that invoke the vertical scrolling action. For horizontal scrolling sideways pushes of a wheel (if it has that) produce transitions on `PAD4` and `PAD5`, which map to interrupt-characters 158 and 159. (156-159 are the highest right-panel characters of character-set 0 that correspond to left-panel control characters, so typically have no other conflicting meaning.)

`(ENABLEWHEELSCROLL NIL)` causes `PAD1`, `PAD2`, `PAD4`, and `PAD5` to be ignored.

Current negative features:

1. When the wheel is depressed for middle-button effect (and no other keys are down), an accidental rotation of the wheel during the transition (up and/or down) may cause unintended scrolling.

We need to develop a strategy, either in Lisp, Maiko, or X, to discriminate intended middle-button pushes from intended scrolling. This is not an issue for track-pad scrolling.

2. When the wheel is rotated over a window that partially occludes a Tedit window with a caret blinking in its unoccluded region, both the target window and the partially obscured Tedit window may scroll.

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 releations 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.....)

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 BACKGROUNDFNS). 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 inconsistant, the number will be preceeded by an asterik ("*"). 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 interactivly 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 indiation 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.

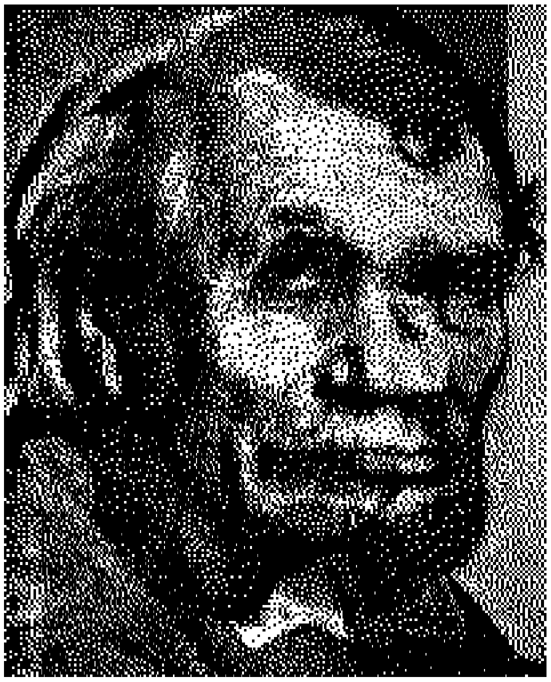
WINK

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

This is a file containing bitmap demos. To bring up a bitmap of Marilyn Monroe winking, type:

(MARILYN)

This file also has bitmaps EINSTEIN and LINCOLN:



"AL" "ABE"

WORDFNS

By: Ron Kaplan (Kaplan.pa@Xerox.com)
Becky Burwell (Burwell.pa@Xerox.com)

Uses: SETSTRINGLENGTH

This document last edited on August 19, 1988.

INTRODUCTION

WORDFNS is a set of functions for manipulating files of words. There are functions to do the following: sort files, manipulate sorted files, provide common i/o functions for word files, provide mapping and translation mechanisms, provide common translation functions, and provide packaged mapping utilities.

The idea behind the mapping mechanism is that you can translate a file or list of files by specifying a read function to operate on each chunk of a file (the obvious two chunks are words and lines). You can specify file specific translation functions, default functions (when file specific functions are not provided) and common translation functions for all files. The input to the first translation function is the result of applying the read function to an input stream open on a file. The output of the first translation function is passed as input to the second translation function, etc.

USE

Note: for any file, if NIL or T is specified then the results are printed in the executive window.

Sorting Files

(SORTWORDFILE *IFILES OFILE COMMONTRANSFNS DEFAULTTRANSFNS READFN COMMONCOMPAREFN KEEPDUPLICATES FIELDS SEPARATOR REVERSEORDERFLG FASTFLG*) [Function]

The functions sorts the words on IFILES and stores the result back on OFILE. Th duplicates are eliminated unless *KEEPDUPLICATES* is non-NIL. For a description of the function of the argumentsCOMMONTRANSFNS, DEFAULTTRANSFNS and READFN see the section entitled "Translation Mechanisms". The argument *FIELDS* is used to specify the sorting order. The separator of the fields is specified in *SEPARATOR*. If *REVERSEORDERFLG* is T the result of the sort is reversed. *FASTFLG* set T causes the sort to caches the fields by consing allowing for a quicker sort (but consumes memory).

FIELDS is one of: NIL, a list of field numbers or else a list of one, two or three element lists of the form: FieldNumber Type CompareFn where type is either STRING (the default) or NUMBER. The default comparefn for STRING is ALPHORDER; for NUMBER is NUMORDER [Argument]

SEPARATOR is one of the following: a character string, a bittable, a list of single character atoms or numbers or one of the special atoms WHITESPACE (indicating a space or tab) or the atom TAB. The default is WHITESPACE.

[Argument]

>> should I put NUMORDER and GetNthField here?<<

Note: two related functions, NUMORDER and GetNthField, are described in the miscellaneous section.

Functions for use with sorted files

In each of the following functions:

COMMENTFILE contains the details of the result of the function (for example, the number of strings that were read in from each file) [Argument]

(COMMONSORTEDFILES *file1 file2 ofile COMMENTFILE*) [Function]

Computes the intersection of two sorted files, *file1* and *file2* and the results are stored on *ofile*. The files are read a line at a time. The value is the full name of *ofile*.

(COMPARESORTEDFILES *file1 file2 ofile IMINUS2 2MINUS1*

COMMENTFILE COMMENT) [Function]

The two sorted files, *file1* and *file2*, are compared a line at a time. The common lines are stored on *ofile*. The output is in two colums: the left column for those lines in *file1* that do not exist in *file2* and the right column for those lines in *file2* that do not exist in *file1*. The two flags *IMINUS2* and *2MINUS1* are used to determine how the the comparisons will be performed. If they are not specified they are both assumed to be T thus meaning that the comparison will be performed by subtracting *file2* from file *file1* and *file2* with *file1* substracted. If only one of *IMINUS2* or *2MINUS1* is specified then only the specified one way comparison will be done. *COMMENT* is intended to be a string which, by default, is the string "Comparison". This string is inserted at the top of the file. The value is the full name of *ofile*.

(DIFFSORTEDFILES *FILE1 FILE2 OUTFILE COMMENTFILE*) [Function]

The result of subtracting *FILE1* from *FILE2* is stored on *OUTFILE*. The files are read a line at a time. The value is the full name of *ofile*.

I/O Functions

In the two major read functions, DREADLINE and DREADWORD, the SPACE, CR, LF and ^Z in any character set are interpreted to be the corresponding character in character set zero.

(DREADLINE *stream string skipsemicolons*) [Function]

Words are read from the word-stream *stream*, smashing them into *string*, which grows as needed. Returns NIL at EOF. Skips leading and trailing separators, and if *skipsemicolons* is non-NIL then sequences from ";" to EOL are treated as a composite separator or end-marker. Unlike DREADWORD (desribed later in this section), segments are separated only by EOL, so compounds are not split into components. Note that *stream* must have been set up so that BIN/READCCODE returns NULL on EOF.

(DREADLINESKIPSC *stream string skipsemicolons*) [Function]

Calls DREADLINE with *skipsemicolons* bound to T.

(DREADWORD *stream string*) [Function]

Words are read from the word-stream *stream*, smashing them into *string*, which grows as needed. Returns NIL at EOF. Skips leading and trailing separators, and treats sequences from ";" to EOL as a composite separator or end-marker. Unlike DREADLINE, segments are separated by space as well as EOL, so splits compounds into components. Note that *stream* must have been set up so that BIN/READCCODE returns NULL on EOF.

(INPUTWORDSTREAM *FILE NOPRINT*) [Function]

Returns a stream that is guaranteed to be open for word-reading (e.g. using DREADLINE or DREADWORD) at the beginning of *FILE*. If *NOPRINT* is NIL then the fullname of the file will be output.

(OUTPUTWORDSTREAM *FILE*) [Function]

Returns and opens a stream for the output of words (sequential text) guaranteed closed when reset context is exited and deleted if there is an error.

Translation Mechanisms

Translation mechanisms are supplied to allow great flexibility in translating one or more files which may be in different formats and have unique translations applied to them. To specify how a file is to be read a read function (READFN) can be specified. Two common read functions, described previously, are DREADLINE and DREADWORD.

As mentioned earlier each file can have unique or common translation functions. A translation function is a function which takes two arguments: a string (the input to be translated) and an optional scratch string which can be destructively modified. The output of the translation function is one of the following: a string, the value T or the value NIL. Readers may wish to refer to the LispUsers module SETSTRINGLENGTH. The special value T denotes that the output is the same as the input. A value of NIL means that nothing will be kept.

>> a better name for translation set <<

>> what is the syntax?<<

IFILES is either a single file name, a single translation set, a list of file names or a list of translation sets. Translation sets have the form: (READFN *READFN TRANFN1 TRANSFN2* ...). (Note the first element of the list is the actual atom READFN and the second element [Argument]

READFN is the read function that is used for reading the files. It is passed a stream. Unless specified otherwise, the default read function is DREADWORD. [Argument]

DEFAULTTRANSFNS is a single function or list of functions which is first applied to the first file. What is given to the translation function is determined by what the read function passed. Unless specified otherwise, the default read function is DREADWORD. The result of applying the first function in DEFAULTTRANSFNS is input to the second function in DEFAULTTRANSFNS. This result is then passed on for application to the functions in COMMONTRANSFNS. The special value T denotes that the output is the same as the input. [Argument]

COMMONTRANSFNS [Argument]

DONTPRINT is the argument which decides whether the details of the translation functions will be printed. By default it is NIL meaning that the details will be printed. [Argument]

(TRANSLATEWORDFILE *IFILES COMMONTRANSFNS DEFAULTTRANSFNS READFN DONTPRINT*) [Function]

TRANSLATEWORDFILE produces an output file by translating each word in (possibly a list of) *IFILES* through a translation function. List elements of files are paired with their own idiosyncratic translation function. Otherwise the *DEFAULTTRANSFNS* is used. *COMMONTRANSFNS* are applied to the results of the default or file-specific translations to produce the translation string. If any translation function returns NIL, that string is skipped. A translation function is assumed to be an identity if it returns T, which makes simple predicates easy.

(COLLECTWORDFILE *IFILES COMMONTRANSFNS DEFAULTTRANSFNS READFN DONTPRINT*)[Function]

Returns the list of non-NIL values of functions applied to words in *IFILES*.

(MAPWORDFILE *IFILES COMMONTRANSFNS DEFAULTTRANSFNS MAPFN READFN DONTPRINT*) [Function]

Maps mapping function over words in *IFILES*. Nothing is setup for output.

Packaged Mapping Utilities

(LONGESTWORDS *FILES COMMONTRANSFNS DEFAULTTRANSFNS READFN DONTPRINT*)[Function]

The list of longest translated words in *FILES* is returned.

(SEXPRCOUNT *FILE RDTBL*) [Function]

Returns the number of s-expressions in *FILE* using *RDTBL* to read.

(WORDCOUNT *IFILES COMMONTRANSFNS DEFAULTTRANSFNS READFN DONTPRINT*)[Function]

The total number of translated words in *IFILES* is returned.

(FINDPREFIXES *IFILES OFILE PREFIXES BUTNOT READFN*) [Function]

FINDPREFIXES produces an output file *OFILE* of those strings read by *READFN* from *IFILES* which match at least one prefix in the list of prefix strings *PREFIXES* and do not match any prefixes in the list of prefixstrings *BUTNOT*.

(FINDSUFFIXES *IFILES OFILE SUFFIXES BUTNOT NOCAPS READFN*) [Function]

FINDPREFIXES produces an output file *OFILE* of those strings read by *READFN* from *IFILES* which match at least one suffix in the list of suffix strings *SUFFIXES* and do not match any suffixes in the list of suffix strings *BUTNOT*. If *NOCAPS* is specified then the match succeeds if the string does not have the first letter capitalized.

(FINDSUBSTRINGS *IFILES OFILE SUBSTRINGS READFN*) [Function]

FINDSUBSTRINGS produces an output file *OFILE* of those strings read by *READFN* from *IFILES* which match at least one substring in the list of substrings *SUBSTRINGS*.

Translation Functions

(MIXEDCASEP *W*) [Function]

Returns W if it contains mixtures of uppercase and lowercase characters after the initial character.

(PROPERP *W*) [Function]

Returns T if the first characters of W is uppercase.

(NOTPROPERP *W*) [Function]

Returns T if the first characters of W is not uppercase.

(REVERSESTRING *W STR*) [Function]

Reverses W into STR and returns STR.

Examples

This example will printout all the lines that have either the prefix "re" or "no" but not the prefix "non".

```
(FINDPREFIXES  '{dsk}Myfile T '("re" "no") '("non") (FUNCTION DREADLINE))
```

This example will output to file {Phylum}<Project>Suffixes all the words in the files {dsk}File1 and {dsk}File2 that end in "ion" that do not have the first letter capitalized.

```
(FINDSUFFIXES  '({dsk}File1 {dsk}File2) '{Phylum}<Project>Suffixes "ion" NIL
T (FUNCTION DREADWORD))
```

Miscellaneous Functions

(GETNTHFIELD *STRING N SEPARATOR FIELDTYPE*) [Function]

The Nth field in STRING is returned using *SEPARATOR* as the field separator and the field type is coerced to type *FIELDTYPE*.

N is a simple positive integer [Argument]

SEPARATOR is the same as that for SORTWORDFILE [Argument]

FIELDTYPE is either the atom NUMBER or the atom STRING and indicates how the type that the field should be coerced to. The default *FIELDTYPE* is STRING.

Example

(GETNTHFIELD   "So long and thanks for all the fish" 4 ’WHITESPACE ’STRING)

returns the string "thanks".

(GETNTHFIELD   "Joe Smith/5551212/12 Pleasant Lane/" 2 "/" ’NUMBER)

returns the integer 5551212.

(DCOPYSTRING *W STR*)   [Function]

Copies string *W* into string *STR* and returns *STR*.

(NUMORDER *NUMBER1 NUMBER2*)   [Function]

Returns >>??<<

Example

```
(SETQ  MyString (ALLOCSTRING 1))

(DCOPYSTRING "This is a much longer string" MyString)
```

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 keywords 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.