en·vos
Medley

Artificial Intelligence Systems

Lisp Library Packages

XEROX

# Artificial Intelligence Systems

*Lisp Library Packages*

# LISP LIBRARY MODULES

en·vōs

300006
Medley Release 1.0
September 1988

Address comments to:
ENVOS
User Documentation
1157 San Antonio Rd.
Mountain View, CA 94043
415-966-6200

LISP LIBRARY MODULES

Medley Release 1.0

300006

SEPTEMBER 1988

Copyright © 1988 by ENVOS Corporation.

All rights reserved.

Envos is a trademark of Envos Corporation.

Xerox® is a registered trademark of Xerox Corporation.

Sun® is a registered trademark of Sun Microsystems Inc.

DEC, VAX, VMS, and VT100 are registered trademarks of Digital Equipment Corporation.

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

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

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

Text was written and produced with ENVOS formatting tools using Xerox printers to produce text masters. The typeface is Modern.

# TABLE OF CONTENTS

# LIST OF FIGURES

[This page intentionally left blank]

# PREFACE

The *Lisp Library Modules* manual describes the library modules. These modules can be loaded into your sysout to provide additional functionality to your Lisp environment.

## Organization of This Manual

For ease of reference, the library modules appear in alphabetical order by their software module name. This makes it easy to find the instructions and description of a particular module without having to consult the index or table of contents.

Each module includes a general description of the module; the requirements to run the module; installation instructions; user interface description; functions that are part of the module; information on implementation of the module; any limitations; and examples where appropriate.

The descriptions of many library modules are related to functions, variables and concepts documented in the *Interlisp-D Reference Manual*, or *IRM*. Because of the extensive changes to the *IRM* in this release, any advice to "see the *IRM*" means that you should also check the *Lisp Release Notes* for the latest, most accurate information.

## Conventions

Conventions used in the *Lisp Library Modules* manual include the following:

Names of Interlisp functions, macros and variables are shown in UPPERCASE; their arguments are in *ITALICS*.

Names of Common Lisp functions, macros and variables are shown in lowercase; their arguments are in *italics*.

A backslash (\) character preceeding a function or variable name signifies that it is a property of the system.

Examples are shown in `terminal 10`.

Messages displayed in the prompt window are shown in **bold**.

Revision bars in the right margin indicate information that has been added or modified since the last release.

References to the *Interlisp-D Reference Manual*, or *IRM*, are used throughout this manual.

## References

We recommend that you use the *Lisp Library Modules* manual with the following publications:

*Interlisp-D Reference Manual,* Volumes I through III, Koto Release, 1985.

*Common Lisp, the Language,* by Guy L. Steele Jr., Digital Press, 1984.

*Common Lisp Implementation Notes,* Lyric Release, 1987.

*Kermit: A File Transfer Protocol,* by F. DaCruz, Digital Press, 1987.

*Lisp Documentation Tools,* (includes *"A User's Guide to TEdit" and "A User's Guide to Sketch"),* Lyric Release, 1987.

*Lisp Release Notes,* Medley Release, 1988.

*Medley 1.0-S User's Guide,* Medley Release, 1988.

# INTRODUCTION

This information pertains to the library modules as a whole, as well as the interaction among modules, and contains changes that have occurred in the library modules since the Lyric release. Medley changes are indicated with revision bars in the right margin.

In general:

- There are several new modules and enhancements to existing modules.
- Several modules have been taken out of the *Lisp Library Modules*, and put into the *LispUsers' Modules*.
- Several modules have been taken out of the *LispUsers' Modules*, and put into the *Lisp Library Modules*.

## What You Should Look For

Before running the modules, pay particular attention to the file dependencies section below. Most modules call other files, which then must be accessible to the system, and often you'll also need files that are not loaded automatically.

Load the library modules with the FILESLOAD function, using the module's name without an extension. Most modules on the distribution contain the extensions .LCOM or .DFASL in their names to designate their compiled form. The exceptions are ETHERRECORDS and SYSEDIT, which are currently distributed in source form. We have attempted to include the correct extension in the descriptions of each module contained in this manual. However, (FILESLOAD...) normally loads the correct files, regardless of their form.

## Modules that are New, Moved, or Replaced

### Modules Moved from the Library to LispUsers

Big
BitMapFns
BusExtender
BusMaster
CirclPrint
CheckSet
CompileBang
Color
C150Stream
DECL
DInfo

FileCache
HelpSys
Iris
LambdaTran
PCallStats
ReadAIS

## Modules Moved from LispUsers to the Library

Cash-File
Hash-File
SysEdit
TableBrowser

## Modules Moved to Their Own Manuals

TEdit
Sketch
CML, CMLArray, CMLArrayInspector (part of Xerox Common Lisp)

## Modules Moved From the Sysout Into the Library

DEdit
Masterscope
Match
Press

## Modules Moved From the Library Into the Sysout

IconW
FreeMenu

## Modules Replaced

Old: FX-80stream, FastFX-80stream, FXprinter
New: FX-80Printer

Old: WhereIs
New: Where-Is

## New Modules

SysEdit
TableBrowser
TextModules

## Details of Changes

The following modules were significantly changed or added to the library with the Medley Release.

### 4045XLPStream

Enabled its graphics capabilities; added 1108 cable/connector pin-outs.

---

A new function has been added to allow owners of the international 4045 (non-USA model) to use the 4045XLPStream software.

(4045XLP.CHANGE.MODE mode) [Function]

This function changes the internal parameters of the software to allow printing on A4 paper with the international fonts. Mode is a string, either "USA" or "INTERNATIONAL", with the default being "USA". Do not use this function unless you have the international font set and A4 paper tray on a non-USA 4045. A4 page size is 2475 pixels wide by 3525 pixels high in portrait, and 3525 x 2475 in landscape mode.

### Cash-File

The new library module Cash-File was formerly in LispUsers. Cash-File is a front end to Hash-File which uses a hash table to cache accesses to hash files. This can provide a significant performace improvement in applications which access a small number of keys repeatedly. For example, the Where-Is library module uses this module to achieve acceptable interactive performance.

### Centronics

Added cable/connector pin-out.

### Chat

Added information about EMACS.

### CopyFiles

When told to copy to a non-existent NS subdirectory, it now asks if it should create it.

### DataBaseFns

Clarifications in the documentation of LOADDBFLG and SAVEDBFLG are included in Medley.

## EditBitMap

Added a description of its user interface.

## FileBrowser

Added enhanced features to Load, Compile, Edit; it now preserves path name of source files when copying to another machine or user, sorts files by attributes, and prints hard copies of directory listings.

The FB command now ignores the package of the attributes you optionally specify, so you can easily use it from a non-Interlisp exec.

The enclosing *'s are now included with the names of the variables *EDITMODE* and *DEFAULT-CLEANUP-COMPILER* .

---

In addition to having outstanding problems fixed, FileBrowser has several new features and NS enhancements.

New features:

- There is an Abort button available during any operation of indefinite duration.
- You can scroll or reshape a FileBrowser that is "busy", e.g, while doing a Recompute.
- The browser title includes a timestamp of when the browser contents were last Recomputed.
- There is a new subcommand of See, "FileBrowse", which opens a FileBrowser on the selected subdirectory. This replaces the odd functionality of the old See/Edit commands that assumed that any file with null name and extension must be a directory; those commands now always treat the selection as a file. FileBrowse is mainly useful in the following situations:
  - When browsing NS directories with depth set finite, or when browsing the top level of a server, which is automatically depth 1.
  - When browsing on Unix, a device that gives Lisp no indication of whether a filename is a directory or not.
- There is a subcommand of Recompute, "Set Depth" that can be used to set the enumeration depth for future recomputes and recursive FileBrowses. You can also set the depth in an FB command by appending :DEPTH n to the command line, e.g., FB "{Pogo:}<Carstairs>" :DEPTH 1.

  The depth counts levels of directories below the last directory in the pattern not containing a wildcard; depth 1 means just the immediate descendants of that directory. Depth is ignored for nontrivial patterns, i.e., anything but "*.*".

- Another new subcommand of Recompute, "Shape to Fit", widens or narrows the browser so that all fields, and no more, are visible but not wider than the screen.
- Directory items are now displayed like files, e.g., you'll see a single line

  Lisp>

  rather than the double line

  <Carstairs>Lisp>

  NIL

  In addition, the "page" count for a directory item is now the total page size of the directory subtree rooted there.
- FileBrowser consumes somewhat less storage now, and there have been some performance improvements, especially for very large browsers.

## FTPServer

FTPServer now supports DELFILE.

---

The Medley Release fixes several bugs in Lisp's handling of PUP FTP connections relating to password handling and filename recognition.

## FX-80Driver

New software, new text, and 1108/1186 cable/connector pin-outs have been added.

---

Comments are now printed in a compressed font.

## GCHax

Documentation contains a new description of the STORAGE function.

## Grapher

Grapher can now print graphs larger than one page. The variable GRAPH/HARDCOPY/FORMAT is used to control the format of the graph when printing to paper. See the function HARDCOPYGRAPH and the variable GRAPH/HARDCOPY/FORMAT in the documentation for Grapher for more information.

A new GRAPH.PROPS field has been added to Graph record, which produces a list in property-list format, and is accessed by the function GRAPHERPROP.

## Hash

Hash is provided for backwards compatability. New applications should use the Medley library module Hash-File instead of this module.

## Hash-File

Hash-File is a new library module, upgraded from LispUsers. Hash-File is similar to but not compatible with the Lyric library module, Hash. Hash-File is modeled after the Common Lisp hash table facility, and Hash was modeled after the Interlisp hash array facility.

## Kermit

Reference to an excellent text/reference book has been added.

## MasterScope

Break when graying a browser has been fixed.

In Medley, MasterScope .LCOM files have been changed to .dfasl file extensions. MasterScope now recognizes Common Lisp structures.

## NSMaintain

The module NSMaintain has been completely revised and has all new documentation. Most commands auto-complete on one or two keystrokes. The Change Password command works again, and there are several new commands for listing objects in the Clearinghouse data base and for manipulating the access lists of groups. There is a more rational set of default inputs offered for most commands, and better feedback is given as to whether a command succeeded or failed.

## RS232

The RS232.TRACE function is now documented in the Medley release.

## Spy

This version of SPY library module works better with Common Lisp and incorporates several new features:

- Enters the pending mode when you bring up the SPY menu by pressing the left or middle button while the control key is down. Any action invoked from the menu is deferred until you next press the left or middle mouse button. For example, you can delete several nodes and then do one update.
- Keeps track of non-symbol frame names.
- Shows the package prefix of symbols in the display.

- Invokes "Merge" menu item from a node menu allowing for a node to merge with its caller.
- Updates SPY.NOMERGEFNS to correspond more closely to "system" functions in Medley.
- Knows about the Medley interpreter.

## TableBrowser

- New functions TB.UNSELECT.ITEM and TB.UNSELECT.ALL.ITEMS fill an inadvertant void in the Lyric version.
- Several off-by-ones in the display algorithms have been fixed.
- Performance on large browsers is improved.
- Clarification of TBAFTERCLOSEFN documentation is included in the Medley release.
- New options to TB.MAKE.BROWSER:
  - The option LINESPERITEM, previously documented but not implemented, is now supported. Alternatively, you can specify explicitly the height of items by giving the options ITEMHEIGHT (total height of each item) and/or BASELINE (the height of the "baseline" relative to the bottom of the item; zero if you don't set it). The BASELINE is used for two things: (1) the *ypos* of the window is set there when the browser's print function is called, and (2) selection and deletion marks are centered between the baseline and the top of the item. Specifying LINESPERITEM is a shorthand for setting ITEMHEIGHT to fontheight*#lines and BASELINE to fontheight*(#lines-1) + fontdescent (i.e., font's baseline for the first line of the item), so that the selection marker, deletion lines, and positioning for printing all point at the first line of a multi-line item. One further difference: If you change the font of the browser, TableBrowser will recompute the height and baseline parameters if you specified LINESPERITEM, but not if you specified ITEMHEIGHT.
  - You can specify an auxiliary window that is to be horizontally scrolled in parallel with the main window by giving the window as the HEADINGWINDOW option. The WIDTH of the window's EXTENT property is maintained in synch with main window. You still need to create the auxiliary window, attach it where you want it and supply it with a REPAINTFN. This is how FileBrowser implements its header line consisting of "Name" and the attribute names.
  - The option LINETHICKNESS specifies how thick to draw deletion lines. It defaults to TB.DELETEDLINEHEIGHT, initially 1. Making it the height of an item gives an alternative "total blackout" method of deletion.

## TCP-IP

Added revised/expanded installation procedure.

---

DIR to VMS via TCP now works.

---

TCP Chat hosts can now be lowercase.

---

(TCPFTP.SERVER) now spawns process and runs in it.

---

TCP-IP to a Sun returns the top-level directory.

---

TCPFTP.DEFAULT.FILETYPES now contains correct entries for LCOM, lcom, DFASL, and dfasl.

---

Files loaded by the high-level modules TCPFTP, TCPFTPSRV, TCPCHAT, and TCPTFTP automatically load their dependencies. If you load files by hand, you must also load their dependencies first. See the section "File Dependencies," or the TCP-IP documentation for more information.

---

There is a new flag:

TCP.ALWAYS.READ.HOSTS.FILE [Variable]

This flag is initially T. Setting it to NIL will cause the system to parse the hosts.txt file only when the filename (stored in the configuration file) is different from the previously read filename, or the write date of the file has changed. The hosts.txt file will always be read at least once when loading the software into a clean sysout.

---

If you change your IP.INIT file while TCP-IP is running, you will be prompted to confirm **Restarting TCP**. In most cases, you should confirm the restart.

## TExec

A TEXEC executive window no longer has GET in the menu of possible actions, since GETting text into an executive window makes no sense.

## TextModules

TextModules is a new library module with the Medley release. It can be used to import and export textfiles and File Manager files. It can bring portable Common Lisp sources into the File Manager without losing any of their contents, and create new textfiles based on the File Manager's description of the textfile contents.

### Virtual Keyboards

The Standard-Russian virtual keyboard now has uppercase Be ( . . . ) and Ve ( . . . ) in the right places.

Loading VirtualKeyboards now adds the item KEYBOARD to the default window menu as well as the background menu. Selecting this item from the default window menu allows you to specify a keyboard for an individual window.

### Where-Is

Where-Is is a new library module, upgraded from LispUsers. This module replaces the Lyric library module WhereIs. This is a new implementation of a facility similar to but not compatible with the Lyric library module WhereIs. Where-Is indexes all definers, but WhereIs only indexed Interlisp FNS definitions.

## File Dependencies

Some modules require that another module be loaded into the sysout in order to run. Automatic dependencies are implemented in the source code, so that the module will load the files it depends on. Contingent dependencies are those files that you may need to load via commands typed into the executive window.

Some modules also depend of specific FONT files. As of this writing the best advice we can give to help you avoid difficulty is that you should make sure all your English font files are loaded in an accessible directory and that your DISPLAYFONTDIRECTORIES and INTERPRESSFONTDIRECTORIES variables are set accordingly.

| **LIBRARY MODULE** | | |
|---|---|---|
| | **AUTOMATIC DEPENDENCIES** (not including system files) | **CONTINGENT DEPENDENCIES** (of module) |
| 4045XLPSTREAM | | DLRS232C or DLTTY or CENTRONICS |
| BROWSER | | MASTERSCOPE |
| | GRAPHER | |
| CASH-FILE | | |
| | HASH-FILE | |
| CHAT | | PUPCHAT or NSCHAT or RS232CHAT or |
| | DMCHAT | TTYCHAT |
| | CHATTERMINAL | and |
| | | DMCHAT (default) or |
| | | TCPCHAT or VTCHAT or TEDITCHAT |
| | | and the corresponding explicit dependencies: |

| | | | | | |
|---|---|---|---|---|---|
| | | | PUPCHAT | | |
| | | | | CHAT | |
| | | | RS232CHAT | | |
| | | | | DLRS232C | |
| | | | | | see "RS232C" below |
| | | | | CHAT | |
| | | | TTYCHAT | | |
| | | | | DLTTY | |
| | | | | | see "RS232C" below |
| | | | | RS232CHAT | |
| | | | | | see above |
| | | | | CHAT | |
| | | | DMCHAT | | |
| | | | | CHATTERMINAL | |
| | | | TCPCHAT | | |
| | | | | see "TCP-IP" below | |
| | | | VTCHAT | | |
| | | | | VT100KP | |
| DATABASEFNS | | | MASTERSCOPE | | |
| DEDIT | | | | | |
| | DEDITPP | | | | |
| EDITBITMAP | | | SCALEBITMAP | | |
| | READNUMBER | | | | |
| FILEBROWSER | | | Printer drivers, fonts | | |
| | TABLEBROWSER | | TEDIT, SEDIT, DEDIT | | |
| FONTSAMPLE | | | FONTSHEETx.IP | | |
| FX-80DRIVER | | | DLRS232C or DLTTY | | |
| GRAPHZOOM | | | | | |
| | GRAPHER | | | | |
| HRULE | | | IMAGEOBJ | | |
| | | | | EDITBITMAP | |
| | | | TEDIT | | |
| KERMIT | | | CHAT | | |
| | | | RS232 or TCP protocols | | |
| | KERMITMENU | | | | |
| | | KERMIT | | | |
| KEYBOARDEDITOR | | | | | |
| | VIRTUALKEYBOARDS | | | | |
| | | see below | | | |
| MASTERSCOPE | | | BROWSER, DATABASEFNS, a Lisp editor | | |
| | MSPARSE | | | | |
| | MSANALYZE | | | | |
| | MSCOMMON | | | | |
| | MS-PACKAGE | | | | |

| | | | | |
|---|---|---|---|---|
| MINISERVE | | | | NS or PUP or XNS |
| NSMAINTAIN | | | | |
| | DES | | | |
| PRESS | | | | FONT.WIDTHS |
| | PUPPRINT | | | |
| RS232 | | | | KERMIT |
| RS232 port : | | | | |
| | DLRS232C | | | |
| | | DOVERS232C | | |
| | RS232CMENU | | | |
| | | DLRS232C | | |
| | | | see above | |
| | | RS232CHAT | | |
| | | | see above | |
| TTY port : | | | | |
| | DLTTY | | | |
| | | DOVERS232C | | |
| | | | see above | |
| | | DLRS232C | | |
| | | | see above | |
| | TTYCHAT | | | |
| | | see CHAT above | | |
| | RS232CMENU | | | |
| | | see above | | |
| SPY | | | | |
| | GRAPHER | | | |
| | READNUMBER | | | |
| | IMAGEOBJ | | | |
| | | EDITBITMAP | | |
| | | | see above | |
| SYSEDIT | | | | MASTERSCOPE |
| | EXPORTS.ALL | | | |
| | | CMLARRAY-SUPPORT | | |
| TCP-IP | | | | |
| | TCP | | | |
| | | TCPLLIP | see below | |
| | TCPCHAT | | | |
| | | TCP | see above | |
| | | CHAT | see above | |
| | TCPCONFIG | | | |
| | TCPDEBUG | | | |
| | | TCP | see above | |
| | TCPFTP | | | |
| | | TCPNAMES | | |

| | | | | |
|---|---|---|---|---|
| | | TCP | see above | |
| | TCPFTPSRV | | | |
| | | TCPFTP | see below | |
| | TCPHTE | | | |
| | TCPLLAR | | | |
| | TCPLLICMP | | | |
| | TCPLLIP | | | |
| | | TCPHTE | | |
| | | TCPLLICMP | | |
| | | TCPLLAR | | |
| | TCPNAMES | | | |
| | TCPTFTP | | | |
| | | TCPUDP | see below | |
| | TCPUDP | | | |
| | | TCPLLIP | see above | |
| TELERAID | | | | |
| | REMOTEVMEM | | | |
| | READSYS | | | |
| | RDSYS | | | |
| | VMEM | | | |
| TEXEC | | | | TEDIT |
| VIRTUALKEYBOARDS | | | | |
| | DANDELIONKEYBOARDS or | | | |
| | DAYBREAKKEYBOARDS or | | | |
| | DORADOKEYBOARDS or | | | |
| | DOVEKEYBOARDS | | | |
| WHERE-IS | | | | |
| | HASH-FILE | | | |

## Additional Notes

DEdit is not error-protected. Doing a ↑ in a DEdit break window closes the DEdit window, too...

In addition, the modules work under all Lisp environments (Interlisp-D, Common Lisp, Xerox Common Lisp). However, many of the functions and variables used within the modules are those of Interlisp-D, and therefore you'll have to make sure that, when you are not in Interlisp, you use the IL: prefix (see the *Release Notes* for more details).

## Koto CML Library Module

If you have files that used the Koto CML library module, with its package-style symbol naming conventions, you will need to convert them to use the correct symbols in Lyric and Medley. The procedure is briefly as follows: see the *Lyric Common Lisp Implementation Notes*, chapter 11, "Reader compatibility feature" for complete details on this mechanism:

First, set the global variable LITATOM-PACKAGE-CONVERSION-ENABLED to T. Then for each of your files, do

(LOAD *file* 'PROP)

(MAKEFILE *file* 'NEW)

Afterwards be sure to set the global variable LITATOM-PACKAGE-CONVERSION-ENABLED back to NIL.

[This page intentionally left blank]

# 4045XLPSTREAM

4045XLPStream implements an image stream for the Xerox 4045 Laser CP, a 300 dot per inch laser printer.

The printer can emulate one of two printers, the Xerox 2700-II laser printer or the Xerox/Diablo 630; the software can produce output for either.

The page size of the 4045 is 2550 pixels wide by 3300 pixels long in portrait mode, and 3300 x 2550 in landscape mode.

There are two communications ports on a workstation that are used most often for printing purposes; the RS232C port and the TTY port. They both use the RS232C standards for data communications, but they differ in that the RS232C port is buffered and, as such, is the preferred port for printing. Depending on the workstation and its options, there may also be a Centronics port available on a system.

4045XLPStream uses one of the corresponding printer (port) driver modules. Output may be sent directly to the 4045 printer via one of these modules, or to a file for printing later.

The 4045XLPStream software has been designed primarily for the RS232 port.

Non-USA model 4045's are supported. See the section "Using the 4045 as a Default Printing Host," for details.

## Requirements

### Hardware

The 1108 or 1109 must have the E-30 option kit installed to be able to use the RS232 port. (If the machine has a port labeled "RS232C" then it has the E-30 kit installed.) The TTY port is labeled "Printer."

On the 1186 the RS232 port is labeled DTE/COMM (it is on the C4 printed circuit board of the workstation), and the TTY port is labeled DCE/Printer on the same board.

The Centronics port is available only on the 1109 (an 1108 with the CPE-FP upgrade).

You'll also need an RS232 or a TTY cable to connect your computer to the printer.

### 4045 PROM and Software Compatibility

630 emulation mode will only work with version 2.1 or higher of the 4045 PROMs. The version of the PROMs in your 4045 can be found by looking at the first number of the Revision number found on the upper-left corner of the configuration sheet.

To receive the latest version of the PROMs, contact the local Xerox Service Representative.

To get the version number of the software, evaluate the variable 4045XLPSTREAM.VERSION and record the value for future reference.

### Software

Make sure the communications module you wish to use is in the currently connected directory, or in one of the subdirectories specified in DIRECTORIES:

RS232 port uses DLRS232C.LCOM
TTY port uses DLTTY.LCOM
Centronics port uses CENTRONICS.LCOM

## Installation

### Software

Load the required .LCOM modules from the library. 4045XLPSTREAM.DFASL should be loaded in the Interlisp Executive. All functions referenced in this document are therefore IL:FN if you are not using the Interlisp Executive.

If you plan to use TEdit, load it BEFORE loading 4045XLPStream, as 4045XLPStream redefines a TEdit function.

Note: Loading 4045XLPStream changes your DEFAULTPRINTINGHOST and DEFAULTPRINTERTYPE to set the 4045 as the default printer. This allows you to use the default hardcopy functions for printing.

### 4045 Emulation Mode Selection

The 4045 printer can normally emulate one of two printers: the Xerox 2700-II laser printer or the Xerox/Diablo 630.

Switch A-2 of the configuration cartridge determines the emulation mode:

On = 630
Off = 2700

Make sure this is the same as the software parameter setting (see below).

### 4045 Port Selection

Set the switches on the 4045 configuration cartridge according to the port you are using. The *Xerox 4045 Laser CP User Manual* gives details about switch settings.

If you are using the RS232 or TTY port, set the switches as follows:

Bank A: switches 1,5,6 on
Bank C: switches 1,4,5 on
Bank B: switches 1,2,3,4,6 on
Bank D: switch 4 on

If the Centronics port is being used, set switch A-1 off.

### 4045 Port Initialization

Upon loading the software, you are prompted for the port you wish to use for printing. The allowable responses are

| | |
|---|---|
| R | RS232 |
| T | TTY |
| C | Centronics |
| S | Server (in which the 4045 is connected to another 1108/86 and the current machine is printing to it via Ethernet), and |
| D | The default port. The software then automatically loads the correct communications module for the port selected. |

For more information on setting the default port, see the notes under "4045XLPStream Options" below.

Note: If you select option S (server), you must set the port by calling

```
(4045XLP.SET.PARAMETERS '((PORT . SERVERPORT))
```

where *SERVERPORT* is the port number of the server. See "Printing Using FTPserver," below.

## Using the 4045 as a Default Printing Host

### Printing Source or TEdit Files

Two system functions are available for sending files to a printer, LISTFILES and SEND.FILE.TO.PRINTER.

LISTFILES (*FILE1 FILE2 FILE3 etc.*) can be used to send a number of files to be printed. As shown here, all files would be sent to the default printer. As described in the *IRM*, this function calls SEND.FILE.TO.PRINTER for each of the files indicated in its list of arguments, and therefore each argument *FILEx* in the list can include values to set the various print options.

(SEND.FILE.TO.PRINTER *FILE HOST OPTIONS*)

*FILE* is a Source or Tedit file.

*HOST* is '.4045XLP.

*OPTIONS* is an A-list of print options: #COPIES, DOCUMENT.NAME, and BREAK.PAGE, which prints a job description page between jobs.

EXAMPLES:

```
(SEND.FILE.TO.PRINTER 'MySourceorTEditFile)
```

Will send the file MySourceorTEditFile to the default printer (4045xlp after loading this module).

```
(SEND.FILE.TO.PRINTER  'MySourceorTEditFile NIL
'(BREAK.PAGE T))
```

Will send the file MySourceorTEditFile to the printer and specifically print a job description page.

```
(SEND.FILE.TO.PRINTER 'MySourceorTEditFile NIL '(#COPIES
2 DOCUMENT.NAME "Another Wonderfully Printed Document
from the 4045"))
```

Will print two copies of the file on the printer attached to the default port with the name given.

## Printing Windows

Select HARDCOPY from the right-button menu of the selected window.

Note: If the 4045XLP is not the default printer, slide off to the right of the HARDCOPY and select TO A PRINTER, then choose 4045XLP. You will only need to do this if DEFAULTPRINTINGHOST has been changed after loading this module.

## Creating 4045XLP Master Files

4045XLP master files are files similar to Interpress master files in that they can be printed without further formatting. They can come from windows, bitmaps, and TEdit/Sketch files. To create a 4045XLP master from a window (including TEdit and Sketch windows), select HARDCOPY from the background menu, slide off to the right and select TO A FILE, and enter the name you wish for the master file with an extension of .4045XLP.

To create a 4045XLP master file for a bitmap, follow these steps:

```
(SETQ FOO (OPENIMAGESTREAM 'MYMASTERFILE.4045XLP))

(BITBLT MYBITMAP 0 0 FOO 0 0)

(CLOSEF FOO)
```

The file MYMASTERFILE.4045XLP can be printed at any time.

Printing a master file is easy. Evaluate

```
(SEND.FILE.TO.PRINTER 'MYMASTERFILE.4045XLP '4045XLP)
```

and it is printed without the delay for formatting.

## Printing via FTPserver

The 4045XLPStream software has been designed to use the FTPserver module to print from remote machines on a local 4045 printer.

In the following example, server is the workstation physically connected to the 4045 and has the PUP Host Number 0#20#; client is connected to server by Ethernet . Both client and server have 4045XLPSTREAM.DFASL loaded.

Note: FTPServer and MiniServe implement simple PUP protocols on the network; they are described elsewhere in this manual.

To get the PUP host number of server, evaluate (PORTSTRING (ETHERHOSTNUMBER)) on server. If the number has not been set yet, the system will prompt you to enter it. Enter an octal number (1-376) that is given to you by your system administrator.

Load FTPSERVER.LCOM onto the server machine. Evaluate (FTPSERVER) on server to start the FTP Watcher process.

From the client you may use the server either as the default printing port, or to copy files to it.

Example:

To make server be the default printer for client, on the client machine evaluate

```
(4045XLP.SET.PARAMETERS (LIST (CONS 'PORT . {0#20#}
LPT:.4045XLP)))
```

Now, any HARDCOPY will automatically be sent and then printed on server without any user intervention.

You can also manually copy master files to the server machine:

```
(COPYFILE 'FOO.4045XLP '{0#20#}LPT:.4045XLP)
```

This will copy the 4045XLP master file to the server machine and automatically print it without any user intervention on the server machine.

Alternatively, when you select HARDCOPY - TO A FILE from the background (right-button) menu, you can enter {0#20#}LPT:.4045XLP, and the file will print immediately after formatting.

## Changing Modes for International 4045s

A function allows owners of the international 4045 (non-USA model) to use the 4045xlpstream software.

(4045XLP.CHANGE.MODE mode) [Function]

This function changes the internal parameters of the software to allow printing on A4 paper with the international fonts. Mode is a string, either "USA" or "INTERNATIONAL", with the default being "USA". Do not use this function unless you have the international font set and A4 paper tray on a non-USA 4045. A4 page size is 2475 pixels wide by 3525 pixels high in portrait, and 3525 x 2475 in landscape mode.

## 4045XLPStream Options

4045XLPStream software implements an image stream interface to the 4045 laser printer. Users do not have to open a stream for the default printing code to work. The default printing code automatically opens the stream and closes it. The methods to access this stream are shown below. For more information on using image streams, see the *IRM*.

There are several options of 4045XLPStream that may be set by the user. Most users will not need to change from the default values. The variable options are stored in the variable 4045XLP.DEFAULTS, which is an instance of the 4045XLP.PARAMETERS record. The following functions will directly modify 4045XLP.DEFAULTS, and thus the option settings.

### 4045 Parameter Names and Values

The following is a list of legal parameter names and values:

SLUG Either an integer 0-255, or NIL; default is NIL. This controls the image to be printed when the stream code sees a character it cannot print. NIL indicates print a slug (black box), a number indicates print the corresponding character.

LANGUAGE Must be either 630 or 2700, default is 2700. This controls the default mode of the stream code. Must be set to the same value as the printer.

The 4045 can implement either 2700 protocols or 630 emulation modes. The mode may also be set when opening the stream by using the MODE option. Make sure switch A-2 on the printer's configuration cartridge corresponds to the mode the software is using.

PORT Must be a valid port for use in printing; default is {RS232}. Controls the default printing port.

Note: May be set to a network address for remote printing. See "Printing Using FTPServer" for more information.

MESSAGESTREAM A window, stream or NIL; default is the prompt window. Controls where messages are sent during the printing process. If it is a window or a stream, messages about the status of the job during printing will be printed to the destination given here. Otherwise the messages are suppressed.

PRINTERRORS A flag, T or NIL; default is NIL. Controls the collection and printing of any printing errors encountered. If it is T, a summary page listing any problems found will be printed at the end of a job.

PRINTHEADER Either a string, T or NIL; default is NIL. Controls the printing of a header page describing the job being printed. T indicates print the page, NIL indicates not to print the page. If it is a string, it indicates to use this string as the title of the header page.

WINDOWTITLE Either a string or NIL; default is NIL. Controls the default title for windows that are printed using the default hardcopy method. If it is a string, the title used will be this string; otherwise "Window Image" will be used as the title.

LANDSCAPE Either T or NIL; default is NIL. Controls the default paper orientation of the stream. T indicates landscape (sideways), NIL indicates portrait orientation.

The orientation may also be changed when opening the stream by using the LANDSCAPE or PORTRAIT options.

Note: If the variable 4045XLP.DEFAULTS is non-NIL before loading 4045XLPStream, the default values will not be initialized to the above settings.

Note: Programmer's note: you can set the default values for all these items by setting 4045XLP.DEFAULTS to an instance of the 4045XLP.PARAMETERS record before loading this module.

## Set Parameters

(4045XLP.SET.PARAMETERS *PARAMETERS*) [Function]

Will set any valid parameter for the 4045. PARAMETERS should be an A-list of the form

```
((PARAMETER . VALUE) (PARAMETER . VALUE) ...)
```

For example, to set the default printing port and the default mode, while leaving all other values as they were, you would evaluate:

```
(4045XLP.SET.PARAMETERS (LIST (CONS PORT '{TTY}) (CONS
LANGUAGE 630)))
```

## Get Parameters

(4045XLP.GET.PARAMETERS *PARAMETERS*) [Function]

Will return the value of any of the parameters listed.

*PARAMETERS* should be a list of the values you wish to check. If *PARAMETERS* is NIL, it will return the current settings of all the options.

For example,

```
(4045XLP.GET.PARAMETERS '(SLUG LANGUAGE PORT))
```

Will return a list similar to the following (depending on the current settings):

```
((SLUG NIL) (LANGUAGE 2700) (PORT {RS232}))
```

## Get, Set Parameters via Inspector Window

An alternate way of showing and changing the parameters for the stream is to call (INSPECT 4045XLP.DEFAULTS) and select AS AN A-LIST from the pop-up menu. This will produce an inspector window with the parameter name on the left and its value on the right. To set a parameter, select it in the inspector window using the left mouse button, then press the middle button and select SET. A window will prompt you for the new value of the parameter selected.

## Examples

4045 image streams are created by the OPENIMAGESTREAM function.

### Opening a 4045 Stream

```
(SETQ 4045STREAM (OPENIMAGESTREAM '{CENTRONICS}.4045XLP))
```

Creates a stream to the 4045 connected to the workstation's Centronics port.

```
(SETQ 4045STREAM (OPENIMAGESTREAM '{RS232} '4045XLP))
```

Creates a stream to the 4045 connected to the workstation's RS232 port.

```
(SETQ 4045STREAM (OPENIMAGESTREAM '{RS232}.4045XLP))
```

Creates a stream to the 4045 connected to the workstation's RS232 port. Notice that the type need only be included as an extension to the port name.

```
(SETQ 4045STREAM (OPENIMAGESTREAM NIL NIL '(MODE 630)))
```

Creates a stream to the default printer (4045) connected to the default port specifically in 630 mode.

```
(SETQ 4045STREAM (OPENIMAGESTREAM NIL '4045XLP '(MODE
2700)))
```

Creates a stream to the 4045 connected to the workstation's default port specifically in 2700 mode.

```
(SETQ 4045STREAM (OPENIMAGESTREAM NIL NIL '(LANDSCAPE
T)))
```

Creates a stream to the defaultprinter (4045) connected to the workstation's default port in LANDSCAPE orientation.

```
(SETQ 4045STREAM (OPENIMAGESTREAM NIL NIL '(PORTAIT T)))
```

Creates a stream to the default printer on the default port in portrait mode.

Note: You must close the stream to make it print. Make sure to close all streams to the 4045 that you open.

## Using a 4045XLP Stream

In the above cases, we set the variable 4045STREAM to a 4045 image stream. Once we have done that, the image stream can thereafter be used as the destination of any graphics operation. Here are some examples of operations that may be performed on a 4045XLP image stream:

```
(BITBLT (WHICHW) NIL NIL 4045STREAM 0 0)
```

Will place an image of the window, in which the cursor is, at position (0,0) on the 4045's page.

```
(DRAWLINE 500 500 3000 500 30 NIL 4045STREAM)
```

Will draw a line 30 spots (1/10 inch) wide from position (500,500) to position (3000,500).

```
(DSPNEWPAGE 4045STREAM)
```

Will cause the current page to be printed and a new one to be started.

Note: This will not have any immediate effect unless you are forcing output immediately to the port. This is unadvisable as it will not allow other tasks to complete until the current stream is closed.

```
(PRINTOUT 4045STREAM "Hello world" T)
```

Will print the string "Hello world" on the 4045's page, followed by a carriage-return/line-feed.

```
(CLOSEF 4045STREAM)
```

Closes the open 4045 image stream and sends the last page to the printer. Make sure you close all open 4045 image streams! If the output was to the default printer and default port, evaluating this will cause the file to be printed.

## Resetting 4045XLPStream

Occasionally, the 4045XLPStream software may become frozen. If this happens, evaluate the function (4045XLP.RESET) which will reset the stream software.

Note: Use this function only when you are positive that nothing else is printing or hardcopying, as it may damage the state of the last job sent.

## Limitations

### 4045 Fonts

The current version of 4045XLPStream does not use fonts other than Titan 10. Documents will print in Titan 10 and Titan 10 Bold only. Source files print in landscape mode, and regular printing in landscape mode is supported.

## Multiple Streams

Do not open multiple streams to the communications ports. It could cause a fatal crash of the port that is connected to the 4045 (ie. for every OPENIMAGESTREAM you did, do a CLOSEF).

## Printing Speed

Using the TTY port for printing large bitmaps or SKETCHs is slow due to the nature of the port.

## Scale factors

SCALEDBITBLT on the 4045 supports scale factors of 1, 2, and 4 only. Since the display screen has a resolution of 72 dots per inch and the printer's resolution is 300, a scale factor of 4 means that the bitmap has the same size on the paper as it has on the screen. Scale factors of 2 and 1 make the printed image proportionally smaller (i.e., 1/2 and 1/4 size, respectively).

## Notecards

If you are using NoteCards, load NOTECARDS-4045XLPPATCH.LCOM.

## TEdit

Be sure to load TEdit before 4045XLPStream, as there is one TEdit function that is redefined by 4045XLPStream.

The LANDSCAPE function in the TEdit Page Layout Menu does not work with this version of software. If you want to print a TEdit document in landscape, proceed as follows:

Change the TEdit margins. To do this, open a Tedit window, GET the file, and bring up the Page Layout Menu. Change the Margins fields to LEFT: 4.5, RIGHT: -10.0, TOP: 19.0, BOTTOM: 3.2 and apply this to all pages (First & Default, Other Left and Other Right).

PUT the modified-for-landscape file and call it LANDSCAPEFILE.TEDIT.

Open an image stream to the printer in landscape mode by evaluating

```
(SETQ MYSTREAM (OPENIMAGESTREAM NIL NIL '(LANDSCAPE T))
```

Do the actual hardcopy by evaluating

```
(TEDIT.HARDCOPY (OPENTEXTSTREAM 'LANDSCAPEFILE.TEDIT)
MYSTREAM NIL NIL NIL '(LANDSCAPE T))
```

After all this is done (there will be a message in the prompt window "Formatting for print... xx pgs done."), you must close the printer file before it will print, so evaluate

```
(CLOSEF MYSTREAM).
```

## SingleFileIndex

If you are using SingleFileIndex, when calling the SINGLEFILEINDEX function, just specify {lpt} as the destination file, instead of {lpt}.4045xlp.

## Sketch

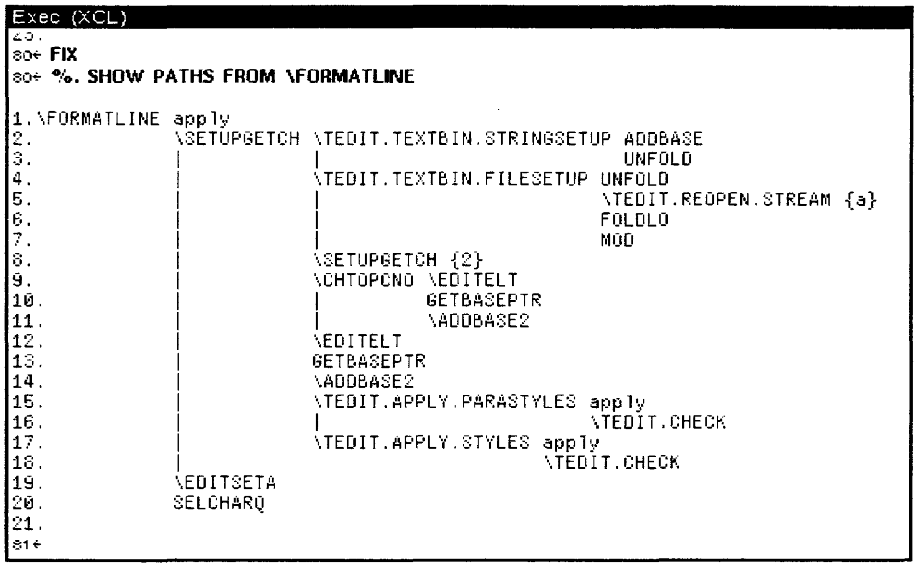
The current version of this module does not support the printing of documents created with the REVERSE feature in Sketch (black/white inversion). There will be a printed output, but not a correct one.

Also, it will not bury a black box by means of a white one; that is, BURY will not erase bits.

[This page intentionally left blank]

Browser modifies the SHOW PATHS command of MasterScope so that the output of the command is displayed as an undirected graph.

Browser makes it easier to use the MasterScope SHOW PATHS command by making it easier to read its results. This is how SHOW PATHS looks before Browser is loaded:

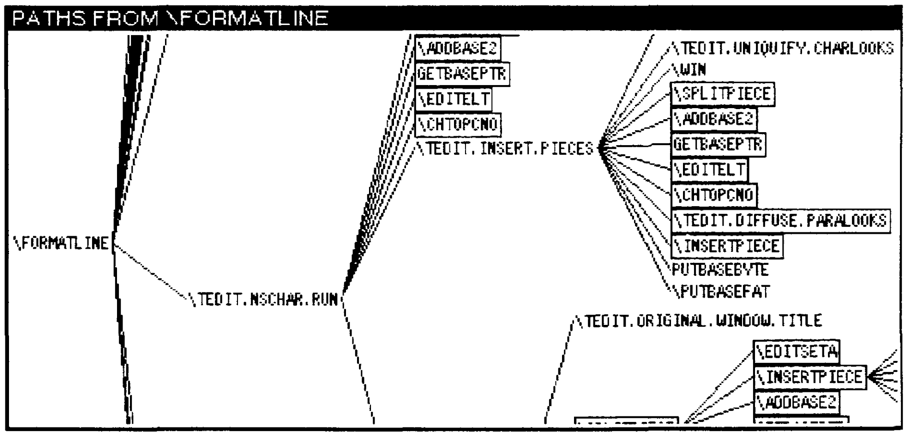


```
Exec (XCL)
29.
80← FIX
80← %. SHOW PATHS FROM \FORMATLINE

1.\FORMATLINE apply
2.          \SETUPGETCH \TEDIT.TEXTBIN.STRINGSETUP ADDBASE
3.          |           |                            UNFOLD
4.          |           \TEDIT.TEXTBIN.FILESETUP UNFOLD
5.          |           |                          \TEDIT.REOPEN.STREAM {a}
6.          |           |                          FOLDLO
7.          |           |                          MOD
8.          |           \SETUPGETCH {2}
9.          |           \CHTOPCNO \EDITELT
10.         |           |         GETBASEPTR
11.         |           |         \ADDBASE2
12.         |           \EDITELT
13.         |           GETBASEPTR
14.         |           \ADDBASE2
15.         |           \TEDIT.APPLY.PARASTYLES apply
16.         |           |                        \TEDIT.CHECK
17.         |           \TEDIT.APPLY.STYLES apply
18.         |                               \TEDIT.CHECK
19.         \EDITSETA
20.         SELCHARQ
21.
81←
```

This shows that FORMATLINE calls SETUPGETCH, which calls EDITELT, etc.

And this is how SHOW PATHS looks after Browser is loaded:


PATHS FROM \FORMATLINE

This shows that FORMATLINE calls TEDIT.NSCHAR.RUN, which calls EDITELT, etc. (on another part of the calling tree).

This window can be shaped and scrolled to see more of the result.

## Requirements

MASTERSCOPE, GRAPHER

## Installation

Load BROWSER.LCOM and the other required modules from the library.

## User Interface

Browser creates a new window for each SHOW PATHS command, but will reuse a window if that window has an earlier instance of the same SHOW PATHS command displayed in it.

The windows can be reshaped and scrolled with the normal window menu commands (which pop up when the right button is pressed in the window title bar).

The windows are active in the sense that nodes in the graph (i.e., functions) can be selected for printing or editing by using the mouse. Clicking with the left button over the name of a function causes that function to be pretty-printed in the Browser printout window.

Selecting the same function again will describe the function, using the MasterScope DESCRIBE command, in the Browser describe window.

Selecting a function with the middle button will call the editor on that function.

The Browser graph is not updated automatically when you edit a function; you must give the SHOW PATHS command again to see the changes.

## Functions

(BROWSER T) turns the Browser on.

The Browser calls LAYOUTFOREST (in Grapher) to generate a graph showing the calling hierarchy.

The format is controlled by the two variables BROWSERFORMAT and BROWSERBOXING (which are set initially to display horizontally and to box functions that occur more than once in the graph).

SHOWGRAPH displays the graph.

The Browser modification to MasterScope can be undone by calling (BROWSER NIL), restoring the teletype-oriented output of SHOW PATHS.

## Limitations

Browser works only with MasterScope.

## Examples

Type the following into an Interlisp Exec window:

```
.ANALYZE ANY ON MY-MODULE
.SHOW PATHS FROM MY-FUNCTION
```

[This page intentionally left blank]

# CASH-FILE

Cash-File is a front end to Hash-File which uses a hash table to cache accesses to hash files. This can provide a significant performance improvement in applications which access a small number of keys repeatedly. For example, the Where-Is library module uses this module to achieve acceptable interactive performance.

Cash-File is similar to but not compatible with the LispUsers' module, HASHBUFFER.

All of the code for Cash-File is in a package called Cash-File. Througout this document Lisp symbols are printed as though in a package which uses the packages Cash-File, Hash-File, and Lisp.

## Installation

Load CASH-FILE.DFASL and HASH-FILE.DFASL from the library.

## Functions

The functional interface is designed to closely resemble that of Hash-File, which was in turn designed to resemble the Common Lisp hash table facility.

(make-cash-file *file-name size cache-size*) [Function]

Creates and returns an empty cash file in *file-name*. *Size* is passed as the *size* argument to `make-hash-file`, while *cache-size* is passed as the *size* argument to `make-hash-table` and determines the maximum number of entries that will be cached.

(get-cash-file *key cash-file &optional default*) [Function]

Just like `get-hash-file` and `gethash`. Retrieves the value stored under *key* in *cash-file* or *default* if there is none. Also returns a second value which is true if a value was found for *key*.

A `setf` method is also defined for `get-cash-file`.

(open-cash-file *file-name cache-size &key direction*) [Function]

Open the existing hash file in *file-name* in *direction* (`:input` or `:io`). *Cache-size* is passed as the *size* argument to `make-hash-table` and determines the maximum number of entries which will ever be cached.

(rem-cash-file *key cash-file*) [Function]

Like `rem-hash-file` and `remhash`. Deletes key from the hash file and the cache. Returns true if and only if there was a value stored under *key*.

(cash-file-p *object*) [Function]

Returns true if and only if *object* is a cash file.

(cash-file-p *object*) ≡ (typep *object* 'cash-file)

(cash-file-hash-file *cash-file*) [Function]

Returns the hash file object which *cash-file* is a front end to.

There are no cash file specific equivalents for `close-hash-file`, `map-hash-file` and `hash-file-count`. For these use the hash file functions on the `cash-file-hash-file`.

## Implementation Notes

A queue is maintained to enable cache deletion when the cache is full. This queue is implemented as a list. Each time a key is accessed, it is moved to the head of the queue. The last element of the queue is deleted when a new key is accessed and the queue is full.

## Limitations

The cache time is not constant but grows linearly with the size of the cache. For this reason, huge caches are not recommended.

# CENTRONICS

The Centronics module implements a stream interface to an industry-standard Centronics printer port. This port is designed to drive Centronics-compatible devices, typically printers. The module allows you to send bytes over the parallel port, and notifies you of any device error conditions.

## Requirements

The Centronics port is found on the Xerox 1109, which is an 1108 equipped with the Extended Processor board (marked CPE FP). It is the upper of the two connectors on the board.

The Centronics cable from the port to the printer should be wired as shown in the Introduction of this manual.

CENTRONICS.LCOM implements a general byte output stream. It is typically used in conjunction with a printer driver module, such as 4045XLPStream, though it can also run by itself.

## Installation

Load CENTRONICS.LCOM from the library.

## User Interface

### Functions

(CENTRONICS.RESET) [Function]

The only user-callable function in the module, it initializes the parallel port and any attached device. It should be called after the printer is powered on.

### Opening a Centronics Stream

To open a stream to the Centronics port, evaluate a form similar to the following:

```
(SETQ CENTRONICS.STREAM (OPENSTREAM '{CENTRONICS}
'OUTPUT))
```

All bytes BOUTed to CENTRONICS.STREAM will be sent to the attached printer. You may only have one stream open to the parallel port at one time; attempts to open others will yield an error.

## Device Errors

When a device error is detected (e.g., printer offline, out of paper, etc.), a break window will pop up.

After resetting the device, type RETURN (the word, not the key) to continue. Type STOP to abort.

## Limitations

The port is available on Xerox 1109 workstations only.

# CHARCODETABLES

CharCodeTables prints character code charts for a given font, showing the characters that exist in a printer in that font. It is useful for illustrating the characters that are available to the user of an application.

Each table is a grid, specific to a font and to the printer to which the output is sent. Across the top are the high-order 8 bits of the character code; down the left are the low-order 8 bits. At 255 of the 256 intersections, a character is printed (code 377 is reserved).

If a particular character doesn't exist on the printer, a black rectangle is shown in its stead.

The tables are printed two to a page, in landscape form. To avoid problems with printer limitations, a group of charts is broken into documents no longer than five pages each for actual printing.

## Xerox Character Codes

The Xerox Character Code Standard specifies a 16-bit character code space containing all the characters in a given font.

For example, there are over 60,000 possible characters in the font Modern 10 Bold; however, many of those character codes haven't yet been assigned. Moreover, a given printer may not have all the characters for a given font that the standard calls for.

The character code space is divided into 255 character sets (numbered 0–376 octal) of 255 characters each (codes 0–377 octal). Character code 377 is reserved as the character-set switching marker in the Xerox file representation of the characters, thereby rendering character set 377 unavailable as well.

Generally, each character set or range of character sets is reserved for a particular use or language. Character set 0, for example, contains the ASCII characters in its lower half, and a variety of common international and commercial symbols in its upper half (corresponding to the 8-bit ISO 6937 standard). Character set 46 is reserved for the Greek alphabet and Greek-specific punctuation marks.

Code assignments are described in detail in:

Xerox System Integration Standard *Character Code Standard*, XNSS 058605, May 1986, version XC1-2-2-0.

## Requirements

The variable INTERPRESSFONTDIRECTORIES must be set to a list of directories which contain font metric files. These files have names of the form

```
{ERIS}<LISP>FONTS>MODERN08-BIR-C#.WD
```

where B = bold, I = italic, and # represents the character set number in octal.

## Installation

Load CHARCODETABLES.LCOM from the library.

## Functions

(SHOWCSETLIST *CSETS FONT*) [Function]

Prints code tables for the character sets in the list *CSETS*, for the given font specification *FONT*.

*CSETS* is a list of one or more numbers that identify the character sets; *FONT* is the name of a font.

(SHOWCSETRANGE *FIRSTCSET LASTCSET FONT*) [Function]

Prints code tables for the character sets from *FIRSTCSET* through *LASTCSET* for the given *FONT*.

(SHOWCOMMONCSETS *FONT*) [Function]

Prints code tables for the most common character sets in the given *FONT*.

(These are character set 0, Greek, Cyrillic, Katakana, Hiragana, and various special symbols)

(SHOWCSET *FONT*) [Function]

Prints code tables for every character set defined in the Xerox Character Code Standard.

## Limitations

This module works only with Interpress fonts.

## Examples

```
(SHOWCSETLIST '(0 238 239) '(MODERN10))
```
or
```
(SHOWCSETLIST '(0 #O356 #O357) '(MODERN10))
```

Prints the code tables for character sets 0, 356 and 357 (octal) that correspond to the Modern 10-point font.

```
(SHOWCSETRANGE 24 26 '(MODERN 10 ITALIC))
```
or
```
(SHOWCSETRANGE #O30 #O32 '(MODERN 10 ITALIC))
```

Prints the code tables for character sets 30 and 32 (octal) and the Modern 10 italic font.

Note: When typing the character set number, remember that the character sets are identified by octal numbers. Therefore you must type either the decimal equivalent of that octal number (e.g., to represent 41 octal, type 33) or the octal number directly, typed as #O41 (where O is the letter O).

[This page intentionally left blank]

# CHAT

Chat is a remote terminal facility that allows you to communicate with other machines while inside Lisp. Chat sets up a Chat connection to a remote machine, so that everything you type is sent to the remote machine, and everything the remote machine prints is displayed in a Chat window.

Chat is an extensible terminal emulation facility. Its core supplies both terminal- and network-protocol- independent functionality; new terminal types and new Chat protocols, based on this core, can be added to Chat at any time. You can choose any terminal type to be used with any network protocol type.

There are currently terminal emulators for the following terminals:

Datamedia 2500

DEC VT100

TEdit (this is actually a TEdit-based Chat window, supporting scrolling and copy-select operations as in standard TEdit).

A number of different network protocol interfaces can be used with Chat. The following protocols are available:

PUP Chat,

NS Chat (using the GAP protocol),

TCP (ARPANET) TELNET,

RS232 Chat (using either the RS232 or TTY ports of the 1108 and 1186 processors).

Each of these is available by loading the corresponding module.

## Requirements

DMCHAT

CHATTERMINAL

One of the network protocols: PUPCHAT or NSCHAT or RS232CHAT or TTYCHAT or TCPCHAT.

One of the terminal emulators: DMCHAT or VTCHAT or TEDITCHAT.

The applicable file dependencies enumerated in the Introduction of this manual.

## Installation

Load CHAT.LCOM from the library.

In addition, you must load at least one of the Chat network protocol modules.

If you want a terminal emulator different from the default DM2500, you must also load it.

## User Interface

Chat prompts for a new window for each new connection. It saves the first window to reuse once the connection in that window is closed (other windows just go away when their connections are closed).

Multiple, simultaneous Chat connections are possible. To switch between typing to different Chat connections, simply press the left button within the Chat window you want to use.

### Opening a Chat Connection

The simplest way to open a Chat connection is to select the CHAT option of the right-button (background) menu. The first time you do this, you will be prompted in the system's prompt window for the name of a host to which to connect. Subsequently, you will be prompted with a menu of all hosts to which you have opened Chat connections; the last entry in this menu will be OTHER, and provides a way for you to connect to new Chat hosts.

The other method of opening a Chat connection is to call the CHAT function directly:

(CHAT *HOST LOGOPTION INITSTREAM WINDOW*) [Function]

Opens a Chat connection to *HOST*, or to the value of DEFAULTCHATHOST. If *HOST* requires login, Chat supplies a login sequence.

You may alternatively specify one of the following values for *LOGOPTION*:

| | |
|---|---|
| Login | Always perform a login. |
| Attach | Always perform an attach (this is likely to be useful only when opening Chat connections to hosts running the Tops-20 or Tenex operating systems). This will fail if you do not have exactly one detached job. |
| None | Do not attempt to log in or attach. |

Note: It is important that you supply information about the types of hosts to which you chat by setting the variable NETWORKOSTYPES (see *IRM*) or DEFAULT.OSTYPE (see *Lisp Release Notes*), as CHAT uses that information to determine whether and how to log in. An incorrect login sequence can inadvertantly expose your password.

If *INITSTREAM* is supplied, it is either a string or the name of a file whose contents will be read as type-in. When the string/file is exhausted, input is taken from the keyboard.

If *WINDOW* is supplied, it is the window to use for the connection; otherwise, you are prompted for a window.

While Chat is in control, all Lisp interrupts are turned off, so that control characters can be transmitted to the remote host. Chat does not turn off interrupt characters until after creating the Chat window, so you can abort the call to Chat by typing control-E while specifying the Chat window region.

If you press the left button in an Executive window, the system's focus-of-attention will be switched to that window. At the same time, keyboard interrupts, such as control-E, will be reenabled. Whenever you select an open Chat window, the focus-of-attention will be returned to the Chat window, and keyboard interrupts will be disabled.

## Chat Menu

Commands can be given to an active Chat connection by pressing the middle mouse button in the Chat window to get a command menu.

Note: The left mouse button, when pressed inside an active Chat window, holds output as long as the button is down. Holding down the middle button coincidentally does this too, but not on purpose; since the menu handler does not yield control to other processes, it is possible to kill the connection by keeping the menu up too long.

CLOSE Closes this connection. Once the connection is closed, control is handed over to the main Lisp Executive window. Closes the Chat window unless it is the primary Chat window.

SUSPEND Same as CLOSE, but always leaves the window open.

NEW Closes the current connection and prompts for a new host to which to open a connection in the same window.

FREEZE Holds type-out from this Chat window. Pressing a mouse button in the window in any way releases the hold. This is most useful if you want to switch to another, overlapping window and there is type-out in this window that would compete for screen space.

DRIBBLE Opens a typescript file for this Chat connection (closing any previous dribble file for the window). You are prompted for a file name. If you want to close an open dribble file (without opening a new one), just type a carriage return.

INPUT Prompts for a file from which to take input. When the end of the file is reached, input reverts to the keyboard.

CLEAR Clears the window and resets the simulated terminal to its default state. This is useful if undesired terminal commands have been received from the remote host that place the simulated terminal into an indeterminate state.

EMACS Turns on or off the Chat EMACS feature, which provides a convenient way to use the workstation's mouse to move the cursor on the remote machine when using the EMACS text editor. When this feature is turned on, pressing the left mouse button in the Chat window causes a sequence of commands to be sent to the remote machine that will cause EMACS to move its cursor to the mouse location.

Use of this feature assumes you know the keystrokes to perform cursor-moving commands; see CHAT.EMACSCOMMANDS if your EMACS does not use the standard ones. Also, it assumes that you are pointing where there is actually text in your document (not white space beyond the end of a line) and that there are no tabs in your text; otherwise, the cursor position may not be where you expect.

RECONNECT In an inactive Chat window, pressing the middle mouse button brings up a menu of one item, RECONNECT, whose selection reopens a connection to the same host as was last in the window. This is the primary motivation for the SUSPEND menu command.

<EMULATOR>MODE The Chat menu also contains a command of this form for each terminal emulator that you have loaded. The <EMULATOR>MODE commands are intended to let you dynamically switch between terminal emulators.

However, this feature is currently defective and should not be used. You must also choose your emulator type, by setting CHAT.DISPLAYTYPES, before opening the Chat connection.

## Customizing Chat

CHAT.DISPLAYTYPES [Variable]

This variable contains a list that assigns the terminal emulators to be used with the hosts. Each entry on the list is of the form:

```
(<HostName><TerminalTypeNumber><TerminalEmulator>)
```

HostName When Chat opens a connection, it scans CHAT.DISPLAYTYPES to find an entry whose HostName field matches the name of the Chat host. If no matching entry is found, it scans the list again, looking for an entry whose HostName field is NIL.

TerminalTypeNumber Is only important when the Chat protocol in use is PUP Chat. This number identifies the terminal type to the Chat host's operating system. Currently, only Tops-20 and Tenex hosts make use of this facility; if the Chat host does not support this feature, the number in the TerminalTypeNumber field is ignored.

TerminalEmulator CHAT uses this field of the entry it finds to choose which terminal type to emulate. Typical terminal emulator names are DM2500, VT100, and TEDIT.

CHAT.KEYACTIONS [Variable]

This variable controls the remapping of the keyboard when the system's focus-of-attention is an active Chat window. The format of this list is:

```
((KEYNAME . ACTIONS) (KEYNAME . ACTIONS) ... )
```

For example, if you prefer the backspace key to send the rubout character (octal 177), you would set CHAT.KEYACTIONS to be:

```
((BS (177Q 177Q NOLOCKSHIFT) . IGNORE))
```

The key actions are assigned when a Chat process is initiated; i.e., changing CHAT.KEYACTIONS will only affect new Chat connections.

CHAT.INTERRUPTS [Variable]

A list of interrupts to pass to INTERRUPTCHAR to assign keyboard interrupts; e.g., ((177Q. HELP)) will cause the DELETE character (code 177) to run the HELP interrupt.

Like CHAT.KEYACTIONS, this variable will only affect new Chat connections.

CHAT.ALLHOSTS [Variable]

A list of host names, as uppercase symbols, to which you desire to chat. Chatting to a host not on the list adds it to the list. These names are placed in the menu used by the background Chat command prompts.

CLOSECHATWINDOWFLG [Variable]

If true, every Chat window is closed on exit. If NIL, the initial setting, then the primary Chat window is not closed.

DEFAULTCHATHOST [Variable]

The host to which CHAT connects when it is called with no HOST argument.

CHAT.FONT [Variable]

If non-NIL, the font used to create Chat windows. If CHAT.FONT is NIL, Chat windows are created with (DEFAULTFONT 'DISPLAY).

Note: Chat fonts must be fixed-width fonts (e.g., Gacha or Terminal) to work well with the DM2500 and VT100 terminal emulators.

CHAT.WINDOW.SIZE [Variable]

This variable is either NIL or a dotted pair of (WIDTH . HEIGHT). The value of the WIDTH field indicates the desired width of the Chat window, in pixels. The value of the HEIGHT field indicates the desired HEIGHT of the window, also in pixels.

CHAT.WINDOW.REGION [Variable]

This variable is either NIL or an instance of a REGION. When CHAT.WINDOW.REGION is non-NIL, its value is used as the region in which to create the first Chat window.

Subsequent windows are created by prompting for the position of a window of CHAT.WINDOW.SIZE dimensions, or, if that variable is NIL, for an arbitrary window region.

CHAT.TTY.PROCESS [Variable]

When you start up CHAT, it takes the TTY immediately if the value is T. (The initial value is T.)

CHAT.EMACSCOMMANDS [Variable]

A list of five character codes; initially the value of (CHARCODE (↑U ↑P ↑N ↑F ↑A)). These character codes are used by the EMACS Argument command in changing the position of the cursor:

Up one line
Down one line
Forward one character
Backward one character
Beginning of line.

CHAT.IN.EMACS? [Variable]

The initial state of the EMACS feature when a Chat connection is started. Initially NIL, meaning the feature is off.

CHAT.PROTOCOLTYPES [Variable]

Each Chat emulator (TTYCHAT, RS232CHAT, PUPCHAT ...) adds an entry onto CHAT.PROTOCOLTYPES which recognizes host names for the appropriate protocol.

For example, loading PUPCHAT adds an entry (PUP . PUPCHAT.HOST.FILTER) and TCPCHAT adds an entry (TCP . TCP.HOST.FILTER).

Site administrators of complex networks may want to reorganize these entries when there are hosts which are running multiple servers, each running different protocols.

## Network Protocols

For the most part, you should not notice too many differences in the behavior of Chat when using one network protocol versus another. The following are unique features of each of the Chat network protocols.

### PUP Chat

PUP Chat is in the file PUPCHAT.LCOM. Implementations of PUP Chat servers exist for Tops-20, Tenex, VAX/Unix, and VAX/VMS operating systems. The PUP Chat protocol contains provisions for automatically setting your terminal type, width, and height whenever you establish a connection or reshape your Chat window.

## NS Chat

The NS Chat protocol (also known as GAP, or Gateway Access Protocol) is used to communicate with hosts running GapTelnet service, including VAX/Unix and the VAX/VMS service XNS/DEC VAX, and also with Xerox 8000-series network services such as 8040 print servers or 8030 file servers. This protocol is contained on the file NSCHAT.LCOM. The NS Chat protocol differentiates among a number of virtual terminal services. When you chat to an NS host, the NS Chat module queries the Clearinghouse for information about the specified host. This information permits the NS Chat module to determine which of the following virtual terminal services are appropriate for the host.

The NS Chat module uses a small set of heuristics to choose which virtual terminal service to invoke, based on information returned by the Clearinghouse. If the Clearinghouse information indicates that only one service type is possible, NS Chat opens a connection to the Chat host and invokes the proper virtual terminal service.

If the Clearinghouse returns information indicating that more than one virtual terminal service is supported by the specified host, you are prompted to choose a service from a menu of the possible service types.

If NS Chat guesses an incorrect service type, or you choose an incorrect service type, you will be prompted to choose a service from a menu of all known virtual service types. If this fails, NS Chat will abandon its attempts to connect to the specified host.

### Remote System Administration

This service lets you log onto print servers and clearinghouse servers, and issue appropriate commands. NS Chat will automatically choose this service when the specified host is registered in the Clearinghouse as any type of server machine.

### Remote System Executive

This service is currently supported by VAX/VMS systems running XNS/DEC VAX, by Unix systems running GapTelnet service, by Lisp workstations running CHATSERVER from the library, and by XDE workstations.

### Interactive Terminal Service

The ITS is a TTY-based interface to NS mail.

### External Communication Service

The External Communication Service (ECS) enables Chat connections to external hosts accessible only by use of a modem. When you open a Chat connection to an ECS, you will be prompted for a telephone number; the ECS will dial that number and complete the connection if a compatible modem answers.

ECS hosts typically support a variety of modem connection characteristics (specific combinations of parity, character length, baud rate, and flow control settings). Each connection type is known by a different Chat host name; check with your system administrator to determine the Chat host name you should use to connect to a particular external host.

### TCP Chat

TCPCHAT.LCOM is the interface to the TCP-based TELNET protocol, which is the protocol in use throughout the ARPANET. It will load and initialize the TCP-IP module, if necessary. Users are encouraged to read the TCP-IP module in this manual.

### RS232 Chat

RS232 Chat is contained on the files RS232CHAT.LCOM and TTYCHAT.LCOM. RS232 Chat enables use of the 1108, 1185, and 1186 RS232 ports; TTY Chat enables use of the 1108, 1185, and 1186 TTY ports. Users should read the RS232 module in this manual.

## Terminal Emulators

### DM2500 Chat

The Datamedia 2500 terminal emulator is contained in DMCHAT.LCOM. To use it, load DMCHAT.LCOM and add entries to CHAT.DISPLAYTYPES in the form:

```
(<HOSTNAME> <TERMINALTYPENUMBER> DM2500)
```

### VT100 Chat

The VT100 emulator is contained in VTCHAT. To use it, load VTCHAT.DFASL and add entries to CHAT.DISPLAYTYPES in the form:

```
(<HOSTNAME> <TERMINALTYPENUMBER> VT100)
```

Currently, the VT100 emulator does not emulate the following features of the actual Digital VT100 terminal:

Dual-width and/or dual-height characters

Graphics character set

Remotely initiated switching between 80- and 132-column mode.

### TEdit Chat

TEdit Chat supplies a "glass TTY" terminal emulator with a TEdit stream storing all characters received during the Chat session. As a result, you can scroll back and forth through a transcript of your session, and you can use the standard TEdit copy-select command to copy blocks of characters from the Chat window to another TEdit window, a Lisp Executive, etc.

To use TEdit Chat, load TEDITCHAT.LCOM, and add entries to CHAT.DISPLAYTYPES in the form:

```
(<HOSTNAME> <TERMINALTYPENUMBER> TEDIT).
```

Note that since TEdit already uses the middle mouse button, you must click in the window's title bar in order to get the usual Chat menu.

[This page intentionally left blank]

# CMLFLOATARRAY

CmlFloatArray implements high-speed floating-point vector and array operations. Although optimized for the case of arrays of element-type single-float, the array operations are generic and will operate on arrays of any element-type.

CmlFloatArray uses special purpose microcode that exploits the full capabilities of the Weitek floating-point chip set, available on 1109s, for doing arithmetic operations on floating-point arrays. On machines without the Weitek floating-point chip set, such as the 1186, these operations will still usually be more efficient than the corresponding scalar implementation.

The functions described here operate on Common Lisp arrays, and may be thought of as extensions of the general Common Lisp sequence functions.

## Requirements

1186 or 1109 (1108 with the extended processor option) and the Weitek floating-point chip set.

## Installation

Load CMLFLOATARRAY.LCOM from the library.

## Functions

(MAP-ARRAY *RESULT MAPFN ARRAY1 ARRAY2. . . ARRAYN*) [Function]

MAP-ARRAY is a general mapping function over arrays and scalars.

Arrays of dimension greater than one are treated as vectors in row-major order; that is, an array A with dimension (2 2) is treated as a vector of length 4 and elements '#(,(aref a 0 0), (aref a 0 1), (aref a 1 0), (aref a 1 1)). All array arguments must be conformable; that is, of the same dimensions.

Scalars (non-arrays) are extended to the common dimension of the other array arguments by copy-on (that is, a scalar is treated as a vector of the appropriate length, each of whose elements is the scalar).

For example, a call to MAP-ARRAY with two arrays of dimensions (4 4) and one scalar and the function MAX as map function will invoke MAX 16 times, with the scalar the third argument in each call.

If *RESULT* is NIL, the map is for effect only (i.e., no array result is returned; MAP-ARRAY is of interest only due to a side effect).

If *RESULT* is a valid element-type, an array of the appropriate dimensionality and element-type will be created to hold the map results.

If *RESULT* is an array, it must be conformable with the other array arguments and it will be side effected by the mapping operation.

*MAPFN* is an arbitrary n-ary Lisp function; i.e., it takes as many arguments as there are arrays passed to MAP-ARRAY. It is unary (takes one argument) if one array is passed to MAP-ARRAY, binary if two arrays are passed, etc.. In the case of unary or binary operations, MAP-ARRAY recognizes certain functions and executes the corresponding operation particularly efficiently.

If the single array argument is of element-type single-float and the result array is of element-type single-float, the following unary operations are recognized and executed in microcode:

| | |
|---|---|
| - (MINUS) | Negates each element of the array argument. |
| ABS | Computes the absolute value of each element of the array argument. |
| TRUNCATE | The single array argument must be of element-type single-float, but the result array may be any element-type which will accomodate the integer results. Truncates (converts to integer, rounding towards zero) each element of the array argument. |
| FLOAT | The single array argument must be of element-type (unsigned-byte 16), and the result array may be of element-type single-float.<br><br>Converts each element of the array argument to a single precision floating point number. |

If both arguments are of element-type single-float and the result array is of element-type single-float, the following binary operations are recognized and executed in microcode.

| | |
|---|---|
| + (PLUS) | Computes the element-wise (element by element) sum of the two arguments. |
| - (MINUS) | Computes the element-wise difference of the two arguments. |
| * (TIMES) | Computes the element-wise product of the two arguments. |
| / (QUOTIENT) | Computes the element-wise quotient of the two arguments. |

(REDUCE-ARRAY *REDUCTION-FUNCTION ARRAY &OPTIONAL INITIAL-VALUE*) [Function]

REDUCE-ARRAY is similar to the sequence function REDUCE but is generalized for arrays of arbitrary dimensionality; that is, the binary mapping function is applied to each element of the single array argument, each time being passed the result of the previous application as well as the current array element. Arrays of dimensionality greater than one are treated as vectors in row-major order. The result of REDUCE-ARRAY is always a scalar.

If *INITIAL-VALUE* is provided, it is used as the starting value of the reduction operation, otherwise the first element of *ARRAY* is the starting value. In the degenerate case of arrays of size zero or one, the use of *INITIAL-VALUE* parallels that of the sequence function REDUCE. REDUCE-ARRAY recognizes certain mapping functions and executes the corresponding operation particularly effeciently.

If the single array argument is of element-type single-float, the following reduction operations are recognized and Executed in microcode.

| | |
|---|---|
| + (PLUS) | Computes the sum of all the array elements. |
| * (TIMES) | Computes the product of all the array elements. |
| MIN | Returns the smallest array element. |
| MAX | Returns the largest array element. |
| MIN-ABS | Returns the smallest array element in absolute value. |
| MAX-ABS | Returns the largest array element in absolute value. |

(EVALUATE-POLYNOMIAL *X COEFFICIENTS*) [Function]

This function calculates the value of a polynomial at the point *X*. The polynomial is described by a vector of coefficients, *COEFFICIENTS*, where *COEFFICIENTS[0]* corresponds to the coefficent of highest degree. If *COEFFICIENTS* is a vector of element-type single-float, then this operation is Executed in microcode.

(FIND-ARRAY-ELEMENT-INDEX *ELEMENT ARRAY*) [Function]

Returns the index of the first element of *ARRAY* that is EQL to *ELEMENT*, or NIL if there is no such element.

## Limitations

This version of CmlFloatArray does not support the FFT functionality of previous versions.

[This page intentionally left blank]

# COPYFILES

CopyFiles makes it easy to copy or move groups of files from one place to another.

## Installation

Load COPYFILES.LCOM from the library.

## Function

(COPYFILES *SOURCE DESTINATION OPTIONS*) [Function]

Copies the files designated by *SOURCE* to the place designated by *DESTINATION.*

*SOURCE* is a pattern such as given to DIRECTORY or DIR; it can also be a list of file names.

*DESTINATION* is either a directory name or a file-name pattern, with a one-to-one match of the wild card characters (*s) in *DESTINATION* to *s in *SOURCE*. The number of *s in each source pattern needs to match the number of *s in each destination pattern. (See examples below.)

*OPTIONS* is a list (if you have only one and it is a symbol, you can supply it as a symbol) that may include one or more of the options specified below.

Note: If the destination is a non-existent NS subdirectory, COPYFILES asks whether it should create it. If you answer YES, then it creates the subdirectory. If you answer NO, it aborts without processing any files.

### *OPTION*: Conversation Mode

You can specify how verbose CopyFiles is about what it is doing:

| | |
|---|---|
| QUIET | Don't print anything while working. |
| (OUTPUT *LISTFILE*) | Print the name of each file that gets copied on *LISTFILE*. (OUTPUT T) is the default. |
| TERSE | Only print a period (.) for each file moved/copied. |

### *OPTION*: Query Mode

You can specify whether CopyFiles should ask for confirmation before each transfer.

| | |
|---|---|
| ASK | Ask each time before moving/copying a file (default is to not ask). |

| | |
|---|---|
| (ASK N) | Ask, with default to No after DWIMWAIT seconds. |
| (ASK Y) | Ask, with default to Yes after DWIMWAIT seconds. |

## *OPTION*: Version Control

CopyFiles normally uses the Lisp function COPYFILE to create a new file. It also usually copies only the highest version, and creates a new version at the destination. Alternatively, you can specify any of the following:

| | |
|---|---|
| RENAME or MOVE | Use RENAMEFILE instead of COPYFILE; i.e., the source is deleted afterwards. |
| ALLVERSIONS | Copy all versions and preserve version numbers. |
| REPLACE | If a file by the same name exists on the destination, overwrite it (don't create a new version). |

Note: When * is used as the source version number, be sure to specify ALLVERSIONS. This is important because some devices list files by version number from highest to lowest, while by default the version numbers at the destination are assigned in ascending order. Hence, if ALLVERSIONS is not specified, the versions may be reversed, as can be verified by looking at the creation dates.

## *OPTION*: When To Copy

CopyFiles normally compares the creation dates of the file on the source and any matching file on the destination to determine whether it is necessary to copy. The following options are mutually exclusive:

| | |
|---|---|
| ALWAYS | Always copy the file. |
| > | Copy only when a file by the same name but an earlier creation date exists on the destination. |
| >A | Similar to >, but also copy if the file doesn't exist on the destination; i.e., > ALWAYS. |
| # | Copy only when a file by the same name but a different creation date exists on the destination. |
| #A | Similar to #, but also copy if the file doesn't exist on the destination, i.e., # ALWAYS. |
| =A | Copy only if there isn't a file of the same name on the destination. |

Not all combinations of options make sense; for example, ALLVERSIONS probably doesn't work right with any date comparison algorithms.

The default setting is (>A); that is, copy the highest version if it doesn't exist on the destination or if an older creation date exists, and print out messages about all files considered.

## *OPTION*: Clean-Up After Copying Files

CopyFiles can be instructed to delete some files after it has finished copying.

| | |
|---|---|
| PURGE | This involves a separate pass (afterwards): any file on the destination which doesn't have a counterpart on the source is deleted. |
| PURGESOURCE | Converse of PURGE (and used by it): if the file is on the source and not on the destination, delete it. |

## Limitations

The creation date comparison does not work when either the source or the destination does not support creation dates. For example, the TCP-IP protocol doesn't support any way to find out the creation date of a remote file. For this reason, COPYFILES can only be used in ALWAYS mode when using a TCP-IP protocol.

## Examples

```
(COPYFILES '{ERIS}<USER>*.MAIL '{PHYLUM}<USER>OLD-*.MAIL)
```

will copy any mail file on {ERIS}<USER> to {PHYLUM}<USER>, copying FOO.MAIL to OLD-FOO.MAIL.

```
(COPYFILES '{ERIS}<USER>*.MAIL '{PHYLUM}<USER>OLD-*.MAIL
'RENAME)
```

will use RENAMEFILE instead.

```
(COPYFILES '({DSK}TEST {DSK}WEST) '{PHYLUM}<MYDIR>)
```

will copy the files TEST and WEST from {DSK} to {PHYLUM}<MYDIR>.

```
(COPYFILES '{PHYLUM}<USER>*.AR '{PHYLEX:}<USER> '=A)
```

will copy all ARs on {PHYLUM}<USER> to the PHYLEX NS file server; if any are already there, it won't bother copying them.

```
(COPYFILES '{PHYLUM}<USER>AR.INDEX '{DSK}AR.INDEX '(>A
REPLACE))
```

will copy the AR index to {DSK}, replacing any older version that is already there.

```
COPYFILES({DSK}*.; {FLOPPY})
```

will copy all files on {DSK} that have no file name extensions to {FLOPPY}.

```
(COPYFILES '{ERIS}<USER> '{PHYLUM}<USER> '(#A PURGE))
```

will make {PHYLUM}<USER> look like {ERIS}<USER>, bringing over any file that isn't already on {PHYLUM} and then deleting the ones that were on {PHYLUM} and aren't on {ERIS} any more.

DataBaseFns makes the construction and maintenance of MasterScope data bases essentially an automatic process.

DataBaseFns modifies the behavior of the Lisp functions MAKEFILE, LOAD and LOADFROM, such that writing out a file also updates and saves a MasterScope data base, and loading the file also loads the data base for you to use.

For example,

```
(LOAD 'FILE1)
```

loads FILE1, then looks for the corresponding data base file FILE1.DATABASE. If the data base exists, it is also loaded. The result is the same as if you had typed:

```
(LOAD 'FILE1)
.ANALYZE ALL ON FILE1
```

The data base will be maintained automatically for any file (containing functions) whose file name has the property DATABASE with value YES. Whenever such a file is dumped via MAKEFILE, MasterScope will analyze any new or changed functions on the file, and a data base for all of the functions on the file will be written on a separate file whose name is of the form FILE.DATABASE. Whenever a file that has a data base property with value YES is loaded via LOAD or LOADFROM, then the corresponding data base file, if any, is also loaded. The data base will not be dumped or loaded if the value of the DATABASE property for the file is NO. The DATABASE property is considered to be NO if the file is loaded with LDFLG = SYSLOAD.

If you change some of the functions defined in FILE1, and perhaps add new ones, then do:

```
(MAKEFILE 'FILE1)
```

Then FILE1 is written out. MasterScope analyzes all changed or new functions and writes FILE1.DATABASE, which contains MasterScope data for all functions on FILE1.

## Requirements

MASTERSCOPE

## Installation

Load MASTERCOPE.DFASL from the library, then load DATABASEFNS.LCOM.

## User Interface

If DataBaseFns is loaded, the first LOAD of a file will ask if you want to load the corresponding data base:

```
(LOAD 'FILE1)
```
**Do you want to load the data base for FILE1?**

And MAKEFILEing a new file will ask if you want to save the data base:

```
(MAKEFILE 'FILE1)
```
**Save the data base for FILE1?**

Once you tell the system YES or NO for a particular file, it will remember and will not ask you again (until you load a new sysout).

## Functions

You can set up default answers to these questions by means of the following variables:

LOADDBFLG [Variable]

This controls whether you are asked before the data base file is loaded:

| | |
|---|---|
| Yes | Always try to load the data base when a file is loaded. |
| No | Never try to load the data base when a file is loaded. |
| Ask | (default) Ask whether to load the data base when a file is loaded. |

SAVEDBFLG [Variable]

This controls whether you are asked before the data base file is saved:

| | |
|---|---|
| Yes | Always try to save the data base when a file is saved. |
| No | Never try to save the data base when a file is saved. |
| Ask | (default) Ask whether to save the data base when a file is saved. |

(DUMPDB *FILE PROPFLG*) [Function]

Dumps a data base for *FILE* then sets the *DATABASE* property to YES, so that data base maintenance for *FILE* will subsequently be automatic.

(LOADDB *FILE ASKFLG*) [Function]

Loads the file FILE.DATABASE if one exists. After the data base is loaded, the *DATABASE* property for *FILE* is set to YES, so that maintenance will be thereafter automatic.

Data base files include the date and full file name of the file to which they correspond. LOADDB will print out a warning message if it loads a data base that does not correspond to the in-core version of the file, and will ask you if you approve.

Note: LOADDB is the only approved way of loading a data base. Attempting to LOAD a data base file will cause an error.

[This page intentionally left blank]

# DEDIT

Many important objects such as function definitions, property lists, and variable values are represented as list structures. There are two list structure editors (SEdit in the system, and DEdit in the library, for backward compatibility) to allow users to modify list structures rapidly and conveniently.

## Description

The list structure editor is most often used to edit function definitions. Editing function definitions in memory is a facility not offered by many Lisp systems, where typically the user edits external text files containing function definitions, then loads them into the environment. In Lisp, function definitions are edited in the environment, and written to an external file using the file manager (see *IRM*), which provides tools for managing the contents of a file.

### History

Early implementations of Interlisp using primitive terminals offered a teletype-oriented editor, which included a large set of cryptic commands for printing different parts of a list structure, searching a list, replacing elements, etc. The library includes an extended, display-oriented version of the teletype list structure editor, called DEdit.

The teletype editor is still available (see *IRM*), as it offers a facility for doing complex modifications of program structure under program control. DEdit also provides facilities for using the teletype editor commands from within DEdit.

### DEdit

DEdit is a structure oriented, modeless, display based editor for objects represented as list structures, such as functions, property lists, data values, etc.

DEdit incorporates the interfaces of the teletype-oriented Interlisp editor, so the two can be used interchangeably. In addition, the full power of the teletype editor, and indeed the full Interlisp system is easily accessible from within DEdit.

DEdit is structure-oriented rather than character-oriented, to facilitate selecting and operating on pieces of structure as objects in their own right, rather than as collections of characters. However, for the occasional situation when character-oriented editing is appropriate, DEdit provides access to the text editing facilities. DEdit is modeless, in that all commands operate on previously selected arguments, rather than causing the behavior of the interface to change during argument specification.

## Requirements

DEDITPP

## Installation

Load DEDIT.LCOM from the library.

Loading DEdit makes it the default Lisp structure editor.

If another Lisp structure editor is already the default and you want to make DEdit the default, call (EDITMODE 'DEDIT) after loading DEdit.

## User Interface

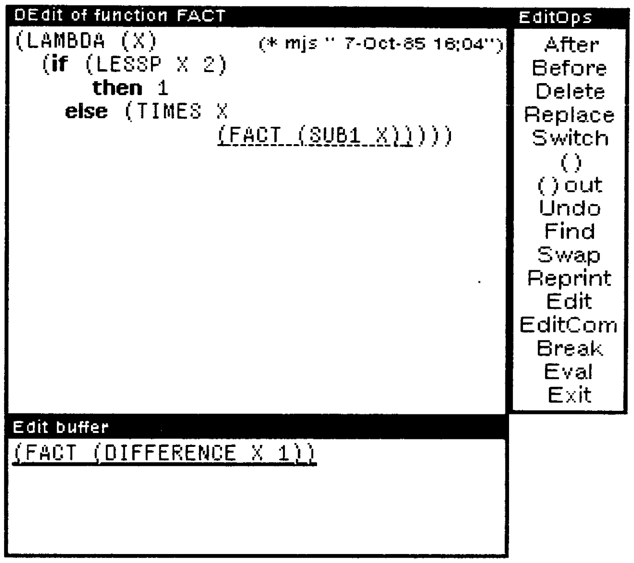
DEdit is normally called using one of the DEdit functions. See also "Advanced Features" below.

### DEdit Window

When DEdit is called for the first time, it prompts for an edit window which is preserved and reused for later DEdits, and it pretty-prints the expression to be edited therein.

Note: The DEdit pretty printer ignores user PRETTYPRINTMACROS because they do not provide enough structural information during printing to enable selection.

The expression being edited can be scrolled by using the standard scroll bar on the left edge of the window. DEdit adds a command menu, which remains active throughout the edit, on the right edge of the edit window. If you type anything, an edit buffer window is positioned below the edit window. This is illustrated in the figure below, which shows the definition of a function called FACT. While DEdit is running, it yields control so that background activities, such as mouse commands in other windows, continue to be performed.



```
(LAMBDA (X)                (* mjs " 7-Oct-85 16:04")
  (if (LESSP X 2)
      then 1
    else (TIMES X
                (FACT (SUB1 X)))))
```

```
(FACT (DIFFERENCE X 1))
```

## Selecting Objects and Lists

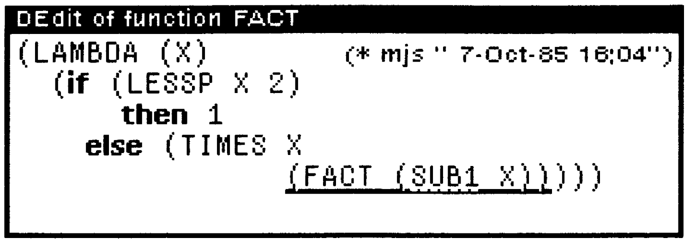
Selection in a DEdit window is as follows:

> The left button selects the object being directly pointed at.
>
> The middle button selects the containing list.
>
> The right button extends the current selection to the lowest common ancestor of that selection and the current position.

The only things that may be pointed at are atomic objects (symbols, numbers, etc) and parentheses, which are considered to represent the list they delimit. White space cannot be selected or edited.

When a selection is made, it is pushed on a selection stack, which will be the source of operands for DEdit commands. As each new selection pushes down the selections made before it, this stack can grow arbitrarily deep, so only the top two selections on the stack are highlighted on the screen. This highlighting is done by underscoring the topmost (most recent) selection with a solid black line and the second topmost selection with a dashed line. The patterns used were chosen so that their overlappings would be both visible and distinct, since selecting a subpart of another selection is quite common.

For example, in the next figure, the last selection is the list (FACT (SUB1 X)), and the previous selection is the single symbol SUB1:



```
(LAMBDA (X)             (* mjs " 7-Oct-85 16:04")
  (if (LESSP X 2)
      then 1
    else (TIMES X
                (FACT (SUB1 X)))))
```

Because you can invoke DEdit recursively, there may be several DEdit windows active on the screen at once. This is often useful when transferring material from one object to another (as when reallocating functionality within a set of programs). Selections may be made in any active DEdit window, in any order. When there is more than one DEdit window, the edit command menu (and the type-in buffer) will attach itself to the most recently opened (or current) DEdit window.

## Typing Characters to DEdit

Characters may be typed at the keyboard at any time. This will create a type-in buffer window which will position itself under the current DEdit window and do a LISPXREAD (which must be terminated by a right parenthesis or a return) from the keyboard. During the read, any character editing subsystem (such as TTYIN) that is loaded can be used to do character level editing on the type-in. When the read is complete, the type-in will become the current selection (top of stack) and be available as an operand for the next command. Once the read is complete, objects displayed in the type-in buffer can be selected from, scrolled, or even edited, just like those in the main window.

You can also enter editing commands directly into the type-in buffer. Typing control-Z will interpret the rest of the line as a teletype editor command that will be interpreted when the line is closed. Likewise, control-S *OLD NEW* will substitute *NEW* for *OLD,* and control-F *X* will find the next occurrence of *X*.

## Copy-Selection

Often, significant pieces of what you wish to type can be found in an active DEdit window. To aid in transferring the keystrokes that these objects represent into the type-in buffer, DEdit supports copy-selection. Whenever a selection is made in the DEdit window with either shift key or the COPY key down, the selection made is not pushed on the selection stack, but is instead unread into the keyboard input (and hence shows up in the type-in buffer). A characteristically different highlighting is used to indicate when copy selection (as opposed to normal selection) is taking place.

Note: Copy-selection remains active even when DEdit is not. Thus you can unread particularly choice pieces of text from DEdit windows into an Exec window.

## Entering DEdit Commands

A DEdit command is invoked by selecting an item from the DEdit command menu. This can be done either directly, using the left mouse button in the usual way, or by selecting a subcommand. Subcommands are less frequently used commands than those on the main edit command menu and are grouped together in submenus under the main menu to which they are most closely related.

For example, the teletype editor defines six commands for adding and removing parentheses (defined in terms of transformations on the underlying list structure). Of these six commands, only two (inserting and removing parentheses as a pair) are commonly used, so DEdit provides the other four as subcommands of the common two.

The subcommands of a command are accessed by selecting the command from the commands menu with the middle button. This will bring up a menu of the subcommand options from which a choice can be made. Subcommands are flagged in the list below with the name of the top level command of which they are options.

If you have a large DEdit window, or several DEdit windows active at once, the edit command window may be far away from the area of the screen in which you are operating. To solve this problem, the DEdit command menu is in an attached window. Whenever the tab key is pressed, the command window will move over to the current cursor position and stay there as long as either the tab key remains down or the cursor is in the command window. Thus, you can pull the command window over, slide the cursor into it and then release the tab key (or not) while you make a command selection in the normal way. This eliminates a great deal of mouse movement.

Whenever a change is made, the pretty-printer reprints until the printing stablizes. As the standard pretty print algorithm is used, and as it leaves no information behind on how it makes its choices, this is a somewhat heuristic process. The REPRINT command can be used to tidy the result up if it is not, in fact, "pretty."

## DEdit Functions

The functions used to start an editor are documented in the "Edit Interface" section of the *Lisp Release Notes*.

(RESETDEDIT) [Function]

Completely reinitializes DEdit. Closes all DEdit windows, so that you must specify the window the next time DEdit is envoked. RESETDEDIT is also used to make DEdit recognize the new values of variables such as DEDITTYPEINCOMS (see "DEdit Parameters," below), when you change them.

## DEdit Commands

All commands take their operands from the selection stack, and may push a result back on the stack. In general, the rule is to select target selections first and source selections second. Thus, a REPLACE command is done by selecting the thing to be replaced, selecting (or typing) the new material, and then selecting the REPLACE command in the command menu.

Using *TOP* to denote the topmost (most recent) element of the stack and *NXT* the second element, the DEdit commands are:

AFTER [DEdit Command]

Inserts a copy of *TOP* after *NXT*.

BEFORE [DEdit Command]

Inserts a copy of *TOP* before *NXT*.

DELETE [DEdit Command]

Deletes *TOP* from the structure being edited. (A copy of) *TOP* remains on the stack and will appear, selected, in the edit buffer.

REPLACE [DEdit Command]

Replaces *NXT* with a copy of *TOP* obtained by substituting a copy of *NXT* wherever the value of the atom EDITEMBEDTOKEN (initially, the & character) appears in *TOP*. This provides a facility like the MBD edit command in Lisp; see EXTRACT, EMBED and IDIOMS below.

SWITCH [DEdit Command]

Exchanges *TOP* and *NXT* in the structure being edited.

() [DEdit Command]

( IN [DEdit Command]

Subcommands of (). Inserts ( before *TOP* (like the LI EDIT command; see "Commands That Edit Parentheses," below).

) IN [DEdit Command]

Subcommand of (). Inserts ) after *TOP* (like the RI EDIT command; see "Commands That Edit Parentheses," below).

() OUT [DEdit Command]

Removes parentheses from *TOP*.

( OUT [DEdit Command]

Subcommand of () OUT. Removes ( from before *TOP* (like the LO EDIT command; see "Commands That Edit Parentheses," below).

) OUT [DEdit Command]

Subcommand of () OUT. Removes ) from after *TOP* (like the RO EDIT command; see "Commands That Edit Parentheses," below).

UNDO [DEdit Command]

Undoes last command.

!UNDO [DEdit Command]

Subcommand of UNDO. Undoes all changes since the start of this call on DEdit. This command can be undone.

?UNDO [DEdit Command]
&UNDO [DEdit Command]

Subcommands of UNDO that allow selective undoing of other than the last command. Both of these commands bring up a menu of all the commands issued during this call on DEdit. When you select an item from this menu, the corresponding command (and if &UNDO, all commands since that point) will be undone.

FIND [DEdit Command]

Selects, in place of *TOP*, the first place after *TOP* that matches *NXT*. Uses the edit subystem's search routine, so supports the full wildcarding conventions of EDIT.

SWAP [DEdit Command]

Exchanges *TOP* and *NXT* on the stack, i.e. the stack is changed, the structure being edited isn't.

SWAP and its subcommands affect the stack and the selections, rather than the structure being edited.

CENTER [DEdit Command]

Subcommand of SWAP. Scrolls until *TOP* is visible in its window.

CLEAR [DEdit Command]

Subcommand of SWAP. Discards all selections (i.e., clears the stack).

COPY [DEdit Command]

Subcommand of SWAP. Puts a copy of *TOP* into the edit buffer and makes it the new *TOP*.

POP [DEdit Command]

Subcommand of SWAP. Pops *TOP* off the selection stack.

REPRINT [DEdit Command]

Reprints *TOP*.

EDIT [DEdit Command]

Runs DEdit on the definition of the atom *TOP* (or CAR of list *TOP*). Uses TYPESOF to determine what definitions exist for *TOP* and, if there is more than one, asks you, via a menu, which one to use. If *TOP* is defined and is a non-list, calls INSPECT on that value. Edit also has a variety of subcommands which allow choice of editor (DEdit, TTYEdit, etc.) and whether to invoke that editor on the definition of *TOP* or the form itself.

Note: DEdit caches each subordinate edit window in the window from which it was entered for as long as the higher window is active. Thus, multiple DEdit commands do not incur the cost of repeatedly allocating a new window.

EDITCOM [DEdit Command]

Allows you to run arbitrary EDIT commands on the structure being DEdited (there are far too many of these for them all to appear on the main menu). *TOP* should be an EDIT command, which will be applied to *NXT* as the current edit expression. On return to DEdit, the (possibly changed) current EDIT expression will be selected as the new *TOP*. Thus, selecting some expression, typing (R FOO BAZ), and selecting EDITCOM will cause FOO to be replaced with BAZ in the expression selected.

In addition, a variety of common EDIT commands are available as subcommands of EDITCOM. Currently, these include ?=, GETD, CL, DW, REPACK, CAP, LOWER, and RAISE.

BREAK [DEdit Command]

Does a BREAKIN AROUND the current expression *TOP*. (See BREAKIN function in *IRM*).

EVAL [DEdit Command]

Evaluates *TOP*, whose value is pushed onto the stack in place of *TOP*, and which will therefore appear, selected, in the edit buffer.

EXIT [DEdit Command]

Exits from DEdit (equivalent to Edit OK; see "Commands For Leaving The Editor," below).

OK [DEdit Command]

STOP [DEdit Command]

Subcommands of EXIT. OK exits without an error; STOP exits with an error. Equivalent to the EDIT commands with the same names.

## DEdit Parameters

There are several global variables that can be used to affect various aspects of DEdit's operation.

EDITEMBEDTOKEN [Variable]

Initially &. Used in both DEdit and the teletype editor to indicate the special atom used as the embed token.

DEDITLINGER [Variable]

Initially T. The default behavior of the topmost DEdit window is to remain active on the screen when exited. This is occasionally inconvenient for programs that call DEdit directly, so it can be made to close automatically when exited by setting this variable to NIL.

DEDITTYPEINCOMS [Variable]

Defines the control characters recognized as commands during DEdit type-in. The elements of this list are of the form (*LETTER COMMANDNAME FN*), where

*LETTER* is the alphabetic character corresponding to the control character desired (e.g., A for control-A),

*COMMANDNAME* is a symbol used both as a prompt and internal tag,

*FN* is a function applied to the expressions typed as arguments to the command.

See the current value of DEDITTYPEINCOMS for examples. DEDITTYPEINCOMS is only accessed when DEdit is initialized, so DEdit should be reinitialized with RESETDEDIT (see "Calling DEdit," above) if it is changed.

DT.EDITMACROS [Variable]

Defines the behavior of the EDIT command when invoked on a form that is not a list or symbol, thus telling DEdit how to edit instances of certain datatypes. DT.EDITMACROS is an association list keyed by datatype name; entries are of the form

(*DATATYPE MAKESOURCEFN INSTALLEDITFN*).

When told to edit an object of type *DATATYPE*, DEdit calls *MAKESOURCEFN* with the object as its argument.

*MAKESOURCEFN* can either do the editing itself, in which case it returns NIL, or else it destructures the object into an editable list and returns that list.

In the latter case, DEdit is then invoked recursively on the list; when that edit is finished, DEdit calls *INSTALLEDITFN* with two arguments, the original object and the edited list. If *INSTALLEDITFN* causes an error, the recursive DEdit is invoked again, and the process repeats until the you either exit the lower editor with STOP, or exit with an expression that *INSTALLEDITFN* accepts.

For example, suppose the you have a datatype declared by (DATATYPE FOO (NAME AGE SEX)). To make sure that instances of FOO can be edited, an entry (FOO DESTRUCTUREFOO INSTALLFOO) is added to DT.EDITMACROS, where the functions are defined by

```
(DESTRUCTUREFOO (OBJECT)
  (LIST (fetch NAME of OBJECT)
        (fetch AGE of OBJECT)
        (fetch SEX of OBJECT)))
(INSTALLFOO (OBJECT CONTENTS)
  (if (EQLENGTH CONTENTS 3)
    then (replace NAME of OBJECT with (CAR CONTENTS))
         (replace AGE of OBJECT with (CADR CONTENTS))
         (replace SEX of OBJECT with (CADDR CONTENTS))
    else (ERROR "Wrong number of fields for FOO" CONTENTS)))
```

## User Interface — Advanced Features

### Multiple DEdit Commands

It is occasionally useful to be able to give several commands at once — either because you think of them as a unit or because the intervening re-pretty-printing is distracting. The stack architecture of DEdit makes such multiple commands easy to construct. You just push whatever arguments are required for the complete suite of commands you have in mind. Multiple commands are specified by holding down the CONTROL key during command selection. As long as the control key is down, commands selected will not be executed, but merely saved on a list. Finally, when a command is selected without the control key down, the command sequence is terminated with that command being the last one in the sequence.

You would rarely construct long sequences of commands in this fashion, because the feedback of being able to inspect the intermediate results is usually worthwhile. Typically, just two or three step idioms are composed in this fashion.

### DEdit Idioms

As with any interactive system, there are certain common idioms on which experienced users depend heavily. In the case of DEdit, many of these idioms concern easy ways to achieve the effects of specific commands from the Edit system, with which many users are already familiar. The DEdit idioms described below are the result of the experience of the early users of the system and are by no means exhaustive. In addition to those that each user will develop to fit his own particular style, there are many more to be discovered and you are encouraged to share your discoveries.

Because of the novel argument specification technique (postfix; target first) many of the DEdit idioms are very simple, but opaque until you have absorbed the "target-source-command" way of looking at the world. Thus, you select where type-in is to go before touching the keyboard. After typing, the target will be selected second and the type-in selected on top, so that an AFTER, BEFORE or REPLACE will have the desired effect. If the order is switched, the command will try to change the type-in (which may or may not succeed), or will require tiresome swapping or reselection. Although this discipline seems strange at first, it comes easily with practice.

Segment selection and manipulation are handled in DEdit by first making them into a sublist, so they can be handled in the usual way. Thus, if you want to remove the three elements between A and E in the list (A B C D E), you select B, then D (either order), then make them into a sublist with the () command. This will leave the sublist (B C D) selected, so a subsequent DELETE will remove it. This can be issued as a single "(); DELETE" command using multiple command selection as described above, in which case the intermediate state of (A (B C D) E) will not show on the screen.

Inserting a segment proceeds in a similar fashion. Once the location of the insertion is selected, the segment to be inserted is typed as a list (if it is a list of atoms, they can be typed without parentheses and the READ will make them into a list, as you would expect). Then, the command sequence "AFTER (or BEFORE or REPLACE); () OUT" (given either as a multiple command or as two separate commands) will insert the type-in and splice it in by removing its parentheses.

Moving an expression to another place in the structure being edited is easily accomplished by a DELETE followed by an INSERT. Select the location where the moved expression is to go to; select the expression to be moved; then give the command sequence "DELETE; AFTER (or BEFORE or REPLACE)". The expression will first be deleted into the edit buffer where it will remain selected. The subsequent insertion will insert it back into the structure at the selected location.

Embedding and extracting are done with the REPLACE command. Extraction is simply a special case of replacing something with a subpiece of itself:

> Select the thing to be replaced.
> Select the subpart that is to replace it.
> REPLACE.

Embedding also uses Replace, in conjunction with the embed token (the value of EDITEMBEDTOKEN, initially the single character atom &). Thus, to embed some expression in a PROG,

> Select the expression.
> Type: (PROG *VARSLST* &)
> REPLACE.

SWITCH can also be used to generate a whole variety of complex moves and embeds.

For example, switching an expression with type-in not only replaces that expression with the type-in, but provides a copy of the expression in the buffer, from where it can be edited or moved to somewhere else.

Finally, you can exploit the stack structure on selections to queue multiple arguments for a sequence of commands. Thus, to replace several expressions by one common replacement, select each of the expressions to be replaced (any number), then the replacing expression. Now hit the REPLACE command as many times as there are replacements to be done. Each REPLACE will pop one selection off the stack, leaving the most recently replaced expression selected. As the latter is now a copy of the original source, the next REPLACE will have the desired effect, and so on.

## Limitations

DEdit is not error-protected. If you select the up-arrow to close a break window which resulted from using the EVAL command, the DEdit window is also closed.

# DISKBACKUP

The DISKBACKUP module provides a way to make a floppy-disk backup copy of the files on a local hard-disk partition. Each file is copied to the floppy disk, then compared with the original.

## Requirements

You will need an 1108/09 or an 1185/86 whose hard disk you wish to back up. You will also need to load the library module COPYFILES, or have it in a place where DISKBACKUP can find it to load it.

You will need to have on hand enough *pre-formatted* floppy disks to hold the entire partition that you intend to save. If you plan to use new, unformatted floppies, please format them before running BACKUP.

## Loading the Module

Load DISKBACKUP.DCOM; it will automatically load COPYFILES if you haven't already loaded it.

## Backing Up Files

The main function is

(BACKUP *VolumeName*) [Function]

*VolumeName* is the name of the logical volume on your hard disk that you want to back up. For example, if you keep your files on the LISPFILES volume (as most people do), you will want to type

```
(BACKUP 'LISPFILES).
```

You will be asked to load floppy disks as the program needs them; after you load each floppy, you will be asked to confirm with the left mouse button. Make sure each floppy is write-enabled.

As each file is copied onto disk, a message is printed showing progress. The "Verifying" message is printed while the copy is being compared to the original; "Done" is printed when the copy has been verified correct. If the file on the rigid disk has an illegal creation date (e.g. date and time were not set when the file was created) then a default date will be used (1-Jan-87 12:00:00) and the message "illegal date" is displayed.

BACKUP first identifies all files which are too large to b contained on a single diskette. Such files are copied to diskettes in HugePilot mode. You will be prompted to insert each diskette during the copy operation. **Be sure to write-enable the diskettes before inserting them in the drive.** When the copy operation for each large file is complete, you will be prompted to place the first floppy in the drive to start the verification step.

Once all large files have been copied and verified, BACKUP proceeds to save files which are small enough to be contained on a single diskette. As each diskette fills up, the system promps you for the next. Each floppy is labelled **Backup Floppy #N**, N = 1, 2, 3...

## Troubleshooting

If the verification step for any file should fail, a break window is opened and an error message is displayed. At this point you have two options: (1) abort the BACKUP process with ↑D and start the disk backup operation from the beginning; or (2) note the name of the file which failed the verification step (for later copying) and up-arrow (↑) out of the break, which resumes the BACKUP process with the *next* file to be saved. The bad copy of the previous file is not deleted from the floppy. If verification fails, it is a good idea to run floppy diagnostics.

Warning: when the verification step fails, open input streams exist for the two files being compared. They should be closed before leaving the break.

## Restoring Files

Files saved with DISKBACKUP can be restored to the local file system in Lyric by using either the COPYFILES.LCOM or FILEBROWSER.LCOM Lyric Library Modules. Be sure the floppy system is in the correct mode: **PILOT** or **HUGEPILOT**, before copying them from floppy.

# EDITBITMAP

EditBitMap provides an interface (EDIT.BITMAP) for creating and editing bitmaps, which may exist as named files or as part of another type of a file (for example, a document written in TEdit).

EditBitMap puts up a menu of bitmap-manipulation commands, one of which is HAND.EDIT, which accesses EDITBM, the Interlisp-D bitmap editor.

EditBitMap also works on cursors (produces new cursor) and symbols (works on the value and resets the value with the result).

## Requirements

READNUMBER
SCALEBITMAP

## Installation

Load EDITBITMAP.LCOM and the required .LCOM modules from the library.

## User Interface

The user interface consists of a function (EDIT.BITMAP), a main operation menu, and a three-part window for low-level pixel editing.

There are two principal ways of entering the bitmap editor. If the bitmap is an object in a document being edited, you can enter the bitmap editor by pressing the left button over the bitmap. If the bitmap is an object you are manipulating as part of a program, you can call the function EDIT.BITMAP from the Executive, passing it the bitmap (typically the value of some variable).

In either case, EditBitMap presents its main menu, from which you select the operation you desire. If the operation is "Hand Edit", EditBitMap brings up a three-part window to show, create, or edit a bitmap. The individual EditBitMap operations can also be performed programmatically or from the Executive (see "Functions").

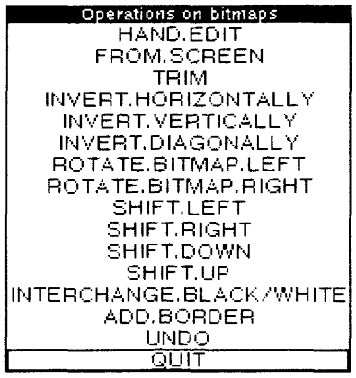
### EDIT.BITMAP

The function EDIT.BITMAP is the principal way to create, view or edit bitmaps stored as the values of variables (or other easily accessible Lisp values):

(EDIT.BITMAP *BITMAP*) [Function]

*BITMAP* may be a bitmap, a cursor, or a symbol. If *BITMAP* is a bit map, then EDIT.BITMAP returns a new bitmap as the result of the edit. If *BITMAP* is a cursor, then EDIT.BITMAP operates on its bitmap and returns a new cursor. If *BITMAP* is a symbol, then EDIT.BITMAP operates on the symbol's value (a bitmap or cursor), and resets the symbol's value to the result of the edit.

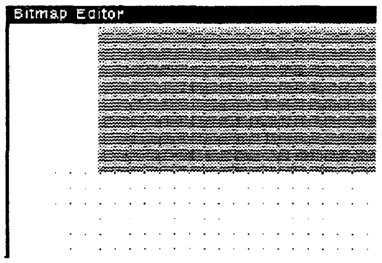
EditBitMap brings up a main menu containing the following items:



EditBitMap performs each command you select, until you select QUIT, at which point it returns the final result of all the edits. You can select UNDO to undo the most recent operation (selecting it several times undoes several operations). If you select HAND.EDIT, you enter the pixel editor EDITBM (see the *IRM*). If you select FROM.SCREEN, EditBitMap prompts you for a screen region from which to initialize a new bit map. The remaining menu items are described under the corresponding "Function" below.

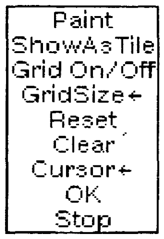
## Window

The EditBitMap window consists of three parts. The main, lower part is the region where the bitmap is displayed on a grid by means of fat pixels. The smaller top part is a gray background against which the middle-button submenu is displayed. The small portion in the upper left corner displays a miniature picture of the entire bitmap.



## Submenu

The EditBitMap submenu is displayed when you press the middle button in the upper gray region of the window. It contains the following items:

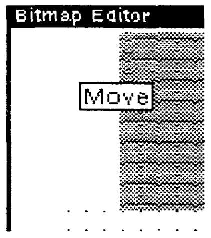


These menu items are described in the system prompt window when an item is selected.

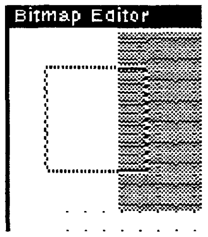
## Mouse Buttons

Pressing the right button anywhere in the EditBitMap window causes the usual window menu to be displayed.

Pressing the left or middle button in the upper left portion of the window presents a MOVE icon.



Pressing on the MOVE icon presents a rectangle which can be moved about in the subwindow. This rectangle indicates the portion of the entire bitmap which will be displayed in the large, bottom subwindow, as soon as the button is released.



Pressing the middle button in the upper, gray subwindow causes the EditBitMap submenu to be displayed (see above).

Pressing the left button in the upper, gray subwindow highlights a rectangle in the upper left subwindow, showing which portion of the entire bitmap is displayed in the large lower subwindow.

Pressing the left button (or dragging the mouse with the left button held down) in the large, lower portion of the window changes the pixels; you are editing at the pixel level.

## Editing an Existing Bitmap

If you have a variable *OLDNAME* whose value is a bitmap, you can make a modified version assigned to *NEWNAME* by typing to the Executive window:

```
(SETQ NEWNAME (EDIT.BITMAP OLDNAME))
```

Or if you want to modify *OLDNAME* in place, pass the quoted name itself:

```
(EDIT.BITMAP 'OLDNAME)
```

In either case, the main menu pops up on the screen. Select the operations you wish to perform, including HAND.EDIT to edit at the pixel level.

Edit the bitmap as needed.

Move the cursor into the gray upper region. Press the middle button to get the submenu. Select OK.

In the main menu, select QUIT. The Executive window displays the new bitmap address.

## Viewing an Existing Bitmap

You can use the hand editor simply to view a bitmap. In this case you don't need to include a SETQ to save the value. For example, type

```
(EDIT.BITMAP BITMAPNAME)
```

Note: Any edits you might be tempted to make while viewing the bitmap in this way will not be saved.

## Creating a New Bitmap

You can use any of the standard graphics interfaces documented in the *IRM* to create a new bitmap. EditBitMap does provide one convenient way to create a bitmap from a region of the screen. In the Executive window, type

```
(SETQ NEWBITMAPNAME (EDIT.BITMAP))
```

Again the main menu pops up on the screen. Select FROM.SCREEN. The cursor changes into the standard region prompt, allowing you to select a region of the screen. Hold down the left mouse button to mark one corner of the region, and drag the mouse to the opposite corner. When you let go of the mouse button, the contents of the screen region you selected are used to initialize a new bitmap. You can then select any other operations you wish to transform this initial image, including HAND.EDIT if appropriate. When finished with all the operations, select QUIT from the main menu. The variable *NEWBITMAPNAME* is now set to the bitmap you have created.

If you want to create a bitmap completely from scratch using the pixel editor, it is simplest to call it directly:

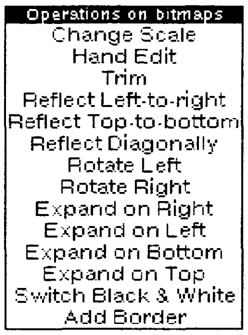
```
(SETQ NEWBITMAPNAME (EDITBM))
```

You will be prompted to supply the width and height of the new bitmap in pixels. When you are finished editing, move the cursor into the gray upper region, press the middle button to get the submenu, and select OK.

If you want to perform further transformations on the new bitmap, edit *NEWBITMAPNAME* as described above for existing bitmaps.

## Editing a Bitmap in a Document

To edit a bitmap that exists inside another file, such as a document being edited in TEdit, press the left or middle button anywhere inside the image of the bitmap. A modified version of EditBitMap's main menu pops up containing the following items:



These menu items correspond exactly to the similarly-named items in EditBitMap's regular menu, except that only one operation is performed at a time (to repeat an operation, just select it again with the mouse; to Undo, use TEdit's Undo command). The menu also contains an additional item, CHANGE SCALE, which allows you to change the scale at which the bitmap's image appears in the document (on the screen and on the printer). Scaling a bitmap changes only the size of its image; it has no effect on its contents (even though on the screen you may not always see it that way).

## Functions

(EDIT.BITMAP *BITMAP*) [Function]

(See above)

(ADD.BORDER.TO.BITMAP *BITMAP NBITS TEXTURE*) [Function]

Returns a new bitmap that is *BITMAP* extended by *NBITS* in all four directions, the border being filled in with *TEXTURE*.

(BIT.IN.COLUMN *BITMAP COLUMN*) [Function]

Returns T if any bit in column numbered *COLUMN* (left = 0) is not 0, NIL otherwise.

(BIT.IN.ROW *BITMAP ROW*) [Function]

Returns T if any bit in row numbered *ROW* (bottom = 0) is not zero, NIL otherwise.

(INVERT.BITMAP.B/W *BITMAP*) [Function]

Returns a new bitmap, which is *BITMAP* with all its bits inverted (black for white).

(INVERT.BITMAP.DIAGONALLY *BITMAP*) [Function]

Returns a new bitmap, which is *BITMAP* flipped about the X = Y diagonal. (The resulting bitmap's width will be *BITMAP*'s height.)

(INVERT.BITMAP.HORIZONTALLY *BITMAP*) [Function]

Returns a new bitmap, which is *BITMAP* flipped about its vertical center line.

(INVERT.BITMAP.VERTICALLY *BITMAP*) [Function]

Returns a new bitmap, which is *BITMAP* flipped about its horizontal center line.

(ROTATE.BITMAP.LEFT *BITMAP*) [Function]

Returns a new bitmap, which is *BITMAP* rotated 90 degrees counterclockwise. (The resulting bitmap's width will be *BITMAP*'s height.)

(ROTATE.BITMAP.RIGHT *BITMAP*) [Function]

Returns a new bitmap, which is BITMAP rotated 90 degrees clockwise. (The resulting bitmap's width will be *BITMAP*'s height.)

(SHIFT.BITMAP.DOWN *BITMAP NBITS*) [Function]

Returns a new bitmap, which is *BITMAP* extended by *NBITS* in the upward direction, the new space being filled in with white.

(SHIFT.BITMAP.UP *BITMAP NBITS*) [Function]

Returns a new bitmap, which is *BITMAP* extended by *NBITS* in the downwards direction, the new space being filled in with white.

(SHIFT.BITMAP.LEFT *BITMAP NBITS*) [Function]

Returns a new bitmap, which is *BITMAP* extended by *NBITS* to the right, the new space being filled in with white.

(SHIFT.BITMAP.RIGHT *BITMAP NBITS*) [Function]

Returns a new bitmap, which is *BITMAP* extended by *NBITS* to the left, the new space being filled in with white.

(TRIM.BITMAP *BITMAP*) [Function]

Returns a new bitmap, which is *BITMAP* trimmed at all four edges of all completely white (0) columns and rows.

(FROM.SCREEN.BITMAP NIL) [Function]

Prompts for a region on the screen and returns a copy of the bitmap.

(INTERACT&SHIFT.BITMAP.LEFT *BITMAP*) [Function]

Prompts for number of bits to shift the *BITMAP* left and returns the new bitmap.

(INTERACT&SHIFT.BITMAP.RIGHT *BITMAP*) [Function]

Prompts for number of bits to shift the *BITMAP* right and returns the new bitmap.

(INTERACT&SHIFT.BITMAP.DOWN *BITMAP*) [Function]

Prompts for number of bits to shift the *BITMAP* down and returns the new bitmap.

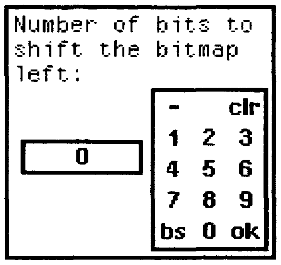
(INTERACT&SHIFT.BITMAP.UP *BITMAP*) [Function]

Prompts for number of bits to shift the *BITMAP* up and returns the new bitmap.

(INTERACT&ADD.BORDER.TO.BITMAP *BITMAP*) [Function]

Prompts for number of bits in the border and calls EDITSHADE to interactively fill in the texture. Returns a new bitmap, which is a bitmap extended in all four directions by the border being filled in with the texture.

Note: If the interactive functions are called from the menus, the prompt for the number of bits is in the form of a ReadNumber window:


Number of bits to shift the bitmap left:

## Limitations

Selecting OK in the submenu does NOT save the edits made in the bitmap. Edits are saved only if you specify a new bitmap name before you begin editing an old one, or if you pass a quoted name to EDIT.BITMAP.

[This page intentionally left blank]

# ETHERRECORDS

EtherRecords contains a collection of record definitions needed for low-level Ethernet programming in Lisp.

## Installation

Load ETHERRECORDS from the library.

## General Purpose Records

ETHERPACKET [Data type]

A data type describing a level-zero Ethernet packet. Use a BLOCKRECORD overlaying this record to define various level-one packets (see PUP and XIP below for examples).

SYSQUEUE [Data type]

A data type implementing a low-level queue for Ethernet use.

QABLEITEM [Record]

A record that overlays any data type whose first field is a pointer used for linking items on a SYSQUEUE.

## NS Records

XIP [Record]

A record overlaying ETHERPACKET describing the layout of a standard Xerox Internet Packet.

ERRORXIP [Record]

A record overlaying ETHERPACKET describing the layout of a standard XNS error packet. The value of the ERRORXIPCODE field of this record is the most interesting one for programmatic handling of XIP errors. The variable XIPERRORCODES contains constants defining most of the standard error codes.

\XIPOVLEN [Constant]

A constant representing the number of bytes in a XIP exclusive of the data portion; i.e., the LENGTH field of a XIP is the byte length of its data portion plus \XIPOVLEN.

\MAX.XIPDATALENGTH [Constant]

A constant, the maximum number of bytes permitted in a standard XIP (546).

NSHOSTNUMBER [Record]

A record describing a 48-bit XNS host number.

NSADDRESS [Data type]

A data type describing a complete XNS address: 32-bit network, 48-bit host, 16-bit socket.

NSNAME [Data type]

A data type describing a standard three-part Clearinghouse name.

## PUP Records

PUP [Record]

A record overlaying ETHERPACKET describing the layout of a standard PUP (PARC Universal Packet).

ERRORPUP [Record]

A record overlaying ETHERPACKET describing the layout of a standard PUP error packet. The value of the ERRORPUPCODE field of this record is the most interesting one for programmatic handling of PUP errors. The variable PUPERRORCODES contains constants defining most of the standard error codes.

PUPADDRESS [Record]

A record describing how to take a 16-bit PUP address apart into 8-bit network and host numbers.

\PUPOVLEN [Constant]

A constant representing the number of bytes in a PUP exclusive of the data portion; i.e., the LENGTH field of a PUP is the byte length of its data portion plus \PUPOVLEN.

\MAX.PUPDATALENGTH [Constant]

A constant, the maximum number of bytes permitted in a standard PUP (532).

\LOCALPUPADDRESS [Macro]
\LOCALPUPHOSTNUMBER [Macro]
\LOCALPUPNETNUMBER [Macro]

These three macros return components of the PUP address of the machine on which the code is running. \LOCALPUPHOSTNUMBER and \LOCALPUPNETNUMBER return the machine's 8-bit host and 8-bit net numbers, respectively; \LOCALPUPADDRESS returns both as a 16-bit number, suitable as a value of the PUPSOURCE field of the PUP record.

# FILEBROWSER

FileBrowser provides a convenient user interface for manipulating files stored on a workstation or file server. It enables you to see, edit, delete, print, load, copy, move, rename, compile, sort, and get several types of information about groups of files. You can also customize FileBrowser by adding your own commands.

## Requirements

TABLEBROWSER

In addition, the HARDCOPY commands require printer drivers and fonts, and the EDIT command requires one or more editors (TEdit, SEdit, DEdit).

## Installation

Load FILEBROWSER.LCOM and the required .LCOM modules from the library.

## User Interface

### Starting FileBrowser

Once you have loaded FileBrowser, there are two ways to open a browser on a set of files:

1. Select the FileBrowser command from the background menu, in which case you are prompted for a file name pattern, or
2. Type the command FB *FILEPATTERN* in your Executive window.

In either case, FileBrowser will prompt you to create a window by presenting you with a dashed rectangle with the mouse cursor and a small geometric design at the lower right corner.

1. Move your mouse until the upper left corner of the rectangle is where you want it on the screen.
2. Hold down the left mouse button and move your mouse down and to the right, thus expanding the window diagonally, until the window is the right size.
3. Release the mouse button. This creates a window group on your screen in the outlined area.

Next, if you did not specify a pattern by using the FB command, FileBrowser prompts you for a file pattern. Type a pattern, as described in the section "Specifying What Files to Browse," below.

FileBrowser enumerates the set of files matching the pattern you requested to see. While the enumeration is in progress, the RECOMPUTE command is grayed out. When the enumeration is finished, you may select files and issue commands. You can scroll the window at any time, even while the browser is busy.

If FileBrowser can't find any files matching the pattern you specified, or you decide you specified the wrong pattern and want to try again, you can specify a new file name pattern from within the browser using the NEW PATTERN command; see RECOMPUTE in the section "FileBrowser Commands," below.

You can have as many active FileBrowsers open at once as you like.

## Specifying What Files to Browse

A full file name in Lisp consists of a device or host (such as your local disk, a file server, or a floppy disk), a principal directory and zero or more subdirectories, a file name (possibly including an extension), and a version number. These fields are put together in the form

```
{HOST}<DIRECTORY>SUBDIRECTORY>FILENAME.EXTENSION;VERSION
```

A file name pattern, as specified to FileBrowser, consists of a file name with one or more pieces omitted or filled with wild cards (*). All the files matching the pattern are listed by FileBrowser. Thus, you can browse all the files in a particular directory, all the files in a subdirectory of that directory, all the files in a directory with a particular extension, and so forth. The wild card * can be used to stand for zero or more consecutive characters in the file name. You can use as many wild cards in a pattern as you wish.

If you leave out some of the fields in a file name pattern, the missing fields are defaulted by the system. Omitted fields in the front of the pattern, i.e., the host, device, or directory fields, are filled in by consulting your connected directory. Other omitted fields are filled in with wild cards unless they are explicitly omitted; i.e., the field is empty, but the preceding punctuation is still present. In more detail, some of the cases are as follows:

If you leave out the name of the host/device, specifying *<DIRECTORY>FILENAME*, FileBrowser will use the name of the host/device for the directory to which you are currently connected.

If you leave out both the device and directory names, specifying *FILENAME*, FileBrowser will use the device and directory to which you are currently connected.

If you do not specify a file name, FileBrowser lists all the files in the specified directory (or the connected directory if you also omitted the host and directory).

If you leave out the extension of a file name, FileBrowser lists all the files with the specified file name and any extension. If you omit the extension but include the period that usually precedes the extension, FileBrowser lists only the files with the specified name and *no* extension.

If you omit the version number of the file name, FileBrowser lists all versions of the matching files. If you omit the version number but include the semicolon that usually precedes the version, FileBrowser lists only the highest version of the matching files.

Thus, the minimal pattern you can type is * (asterisk—enumerate all files in the connected directory) or ; (semicolon—enumerate just the highest version of all files). If you press the RETURN key without giving a pattern, FileBrowser aborts the prompt for a pattern, leaving you with an empty browser in which the only things you can do are change some FileBrowser parameters (see the subcommands of RECOMPUTE in the section "FileBrowser Commands," below) and then use the RECOMPUTE command to be prompted for a pattern again.

### Examples

The pattern * specifies all files in the connected directory. It is equivalent to *.* or *.*;*.

The pattern <FOO>BAR specifies all files in directory FOO with name BAR and any extension. It is equivalent to <FOO>BAR.*;*.

The pattern <FOO>BAR. specifies all files in directory FOO with name BAR and *no* extension. It is equivalent to <FOO>BAR.;*.

The pattern *.TEdit specifies all files in the connected directory with the extension TEdit. It is equivalent to *.TEdit;*.

The pattern *.TEdit; specifies only the newest version of all files in the connected directory with the extension .TEdit.

The pattern <FOO>A*E specifies all files in directory FOO whose names begin with A and end with E and have any extension.

The pattern {TOAST}<FOO>*MY* specifies all files in directory {TOAST}<FOO> whose names contain the substring MY and any extension.

## Using the FileBrowser Window

The FileBrowser window has six major subwindows, which from top to bottom are as follows:

PROMPT window

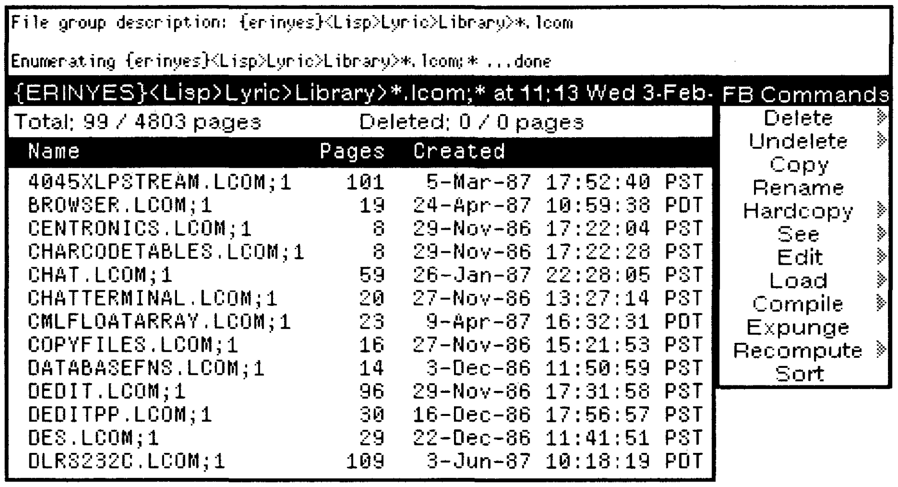
This topmost subwindow is where FileBrowser prints messages about what it is doing and receives input from you. Its contents are cleared before every command.

TALLY window

This subwindow immediately below the prompt window keeps a running tally of the total number of files listed in the window and the number of files that you have marked for deletion. In addition, if one of the attributes you are displaying is a size attribute (Pages or Length, as in the INFO menu, described below), this window maintains a tally of the total number of pages in the files listed and the files marked for deletion.

This window also has a title bar across the top identifying the pattern you specified and the time at which the directory enumeration was performed.

The window is blank while the files are being enumerated.



```
File group description: {erinyes}<Lisp>Lyric>Library>*.lcom

Enumerating {erinyes}<Lisp>Lyric>Library>*.lcom;* ...done
{ERINYES}<Lisp>Lyric>Library>*.lcom;* at 11:13 Wed 3-Feb-
Total: 99 / 4803 pages        Deleted: 0 / 0 pages
 Name                  Pages  Created
 4045XLPSTREAM.LCOM;1    101   5-Mar-87 17:52:40 PST
 BROWSER.LCOM;1           19  24-Apr-87 10:59:38 PDT
 CENTRONICS.LCOM;1         8  29-Nov-86 17:22:04 PST
 CHARCODETABLES.LCOM;1     8  29-Nov-86 17:22:28 PST
 CHAT.LCOM;1              59  26-Jan-87 22:28:05 PST
 CHATTERMINAL.LCOM;1      20  27-Nov-86 13:27:14 PST
 CMLFLOATARRAY.LCOM;1     23   9-Apr-87 16:32:31 PDT
 COPYFILES.LCOM;1         16  27-Nov-86 15:21:53 PST
 DATABASEFNS.LCOM;1       14   3-Dec-86 11:50:59 PST
 DEDIT.LCOM;1             96  29-Nov-86 17:31:58 PST
 DEDITPP.LCOM;1           30  16-Dec-86 17:56:57 PST
 DES.LCOM;1               29  22-Dec-86 11:41:51 PST
 DLRS232C.LCOM;1         109   3-Jun-87 10:18:19 PDT
```

BROWSER window

This is the principal subwindow, in which the files matching the specified pattern are listed. Each file's name appears at the left, and various attributes of the file are displayed in columns to the right. A title bar across the top of the browser window identifies the contents of each column (e.g., Name, Pages, Created). The files are listed in alphabetical order, with multiple versions of the same file listed in decreasing version order; i.e., the newest version appears first. The width of the column listing the file names is initially chosen to be appropriate for average-sized file names. If the files you asked to browse have particularly long names, then when FileBrowser has finished listing all the files it may choose to redraw the browser window with the attribute columns moved farther to the right to accommodate the longer file names.

COMMAND menu

This menu appears vertically along the right side of FileBrowser window (under a title bar "FB Commands") and lists the commands that you may select to perform operations on the files in the browser, or to change the appearance of the browser. Most of the commands operate on the set of currently selected files (see the section "Selecting Files" below). Some commands have subcommands, as indicated by the small triangle alongside them, which can be selected by holding down the left mouse button and sliding the mouse to the right over the triangle.

INFO menu

An additional subwindow, the Info menu, is not normally displayed. It is used to change the set of file information (attributes) displayed in the browser (see the section "Getting Information About Files," below).

SCROLL bar

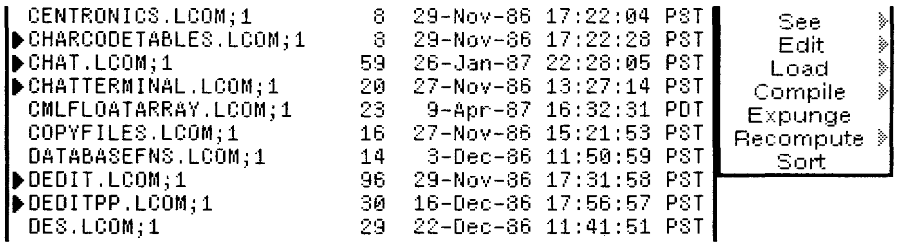
If there are more files in the listing than fit at one time in the browser window, you can scroll the browser window to view more files. Slide the mouse cursor out the left side of the browser window to get the scroll bar and press the left mouse button to scroll the region up and the right mouse button to scroll it down. Pressing a mouse button when the cursor is near the bottom of the scroll bar will scroll the region by larger increments than when the cursor is at the top.

You can also press the middle mouse button in the scroll bar to move the listing to the place that corresponds to that position in the scroll bar. For example, pressing the middle mouse button when the cursor is at the bottom of the scroll bar will display the end of the listing. This quick-scrolling technique is called thumbing. The gray box in the scroll region indicates where the currently displayed contents are, relative to the entire contents of the browser.

Similarly, if there is more attribute information than fits in the browser window, you can scroll the browser window horizontally to view the rest of the attribute information. To do this, slide the mouse out the bottom of the browser window to get the horizontal scroll bar. The left button scrolls to the left, the right button to the right.

## Selecting Files

Most FileBrowser operations are performed by selecting a single file or set of files, then giving a command that specifies what you want to do with the selected files. The current selection is indicated by a small triangle in the left margin of the browser next to each selected file.



```
  CENTRONICS.LCOM;1          8  29-Nov-86 17:22:04 PST
▶ CHARCODETABLES.LCOM;1      8  29-Nov-86 17:22:28 PST
▶ CHAT.LCOM;1               59  26-Jan-87 22:28:05 PST
▶ CHATTERMINAL.LCOM;1       20  27-Nov-86 13:27:14 PST
  CMLFLOATARRAY.LCOM;1      23   9-Apr-87 16:32:31 PDT
  COPYFILES.LCOM;1          16  27-Nov-86 15:21:53 PST
  DATABASEFNS.LCOM;1        14   3-Dec-86 11:50:59 PST
▶ DEDIT.LCOM;1              96  29-Nov-86 17:31:58 PST
▶ DEDITPP.LCOM;1            30  16-Dec-86 17:56:57 PST
  DES.LCOM;1                29  22-Dec-86 11:41:51 PST
```

To select one file, point to any part of the line (which lists the file name and its attributes) and press the left mouse button. If other files are already selected, this unselects them; thus, a file selected with the left mouse button is always the only selection.

To add a single file to the current selection, press the middle mouse button at any place in the line. The file is selected without unselecting any other file.

To remove a single file from the current selection, hold down the control key and press the middle mouse button at any place in the line. The file is unselected without affecting any other file.

To extend the selection to include a group of contiguous files, that is, to select all the files between a file and the nearest already selected file, press the right mouse button on any part of the line. You can only extend the selection from the first selected file upward, or the last selected file downward. In addition, files marked for deletion are not normally selected when you extend.

If you want to include all files, both deleted and undeleted, hold down the control key while extending the selection.

Some lines in a FileBrowser display are directory-only lines. These lines are slightly indented and name the directory and subdirectory to which the files listed below that line belong. You cannot select in these lines, though you can copy-select them (see the section "Copy-Selecting Files," below).

## Commands that Require Input

Some FileBrowser commands require input from you. For example, the COPY command requires that you supply a destination file name. When a command requires input, FileBrowser prints a prompt message in its prompt window. This is usually followed by a default answer. If you want the default answer, you can just press the carriage return to finish the input. If you want to specify a different answer, simply start typing it; the default answer is erased and your answer replaces it.

Alternatively, you can modify the default answer by backspacing over individual letters, or typing control-W to back up over complete words. Typing control-Q erases the entire answer. You can also use the mouse to edit your answer, using the same rules as followed by the Executive (see the documentation of TTYIN). Briefly, the left mouse button positions the caret at a character boundary; the middle mouse button positions the caret at the nearest word boundary; and the right mouse button deletes the characters between the caret and the mouse.

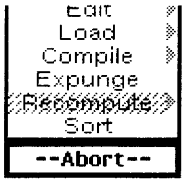
When you have finished, position the caret at the end of your answer, if it isn't there already, and press the carriage return. You can also type control-X to finish your answer even if the caret isn't at the end.

If you change your mind and want to abort the command, supply an empty input; i.e., if there is an answer in progress, backspace over it or type control-Q to erase it, then press the carriage return. FileBrowser prints "aborted" and aborts the command. In most situations, the control-E interrupt can also be used to abort your answer.

While you are typing an answer, you can copy-select file names out of the browser (or any other browser), as described below in the section "Copy-Selecting Files". This can be useful, for example, if you wish to rename a file to a similar name in the same directory, or move a file into a subdirectory listed in the browser.

## Aborting Commands

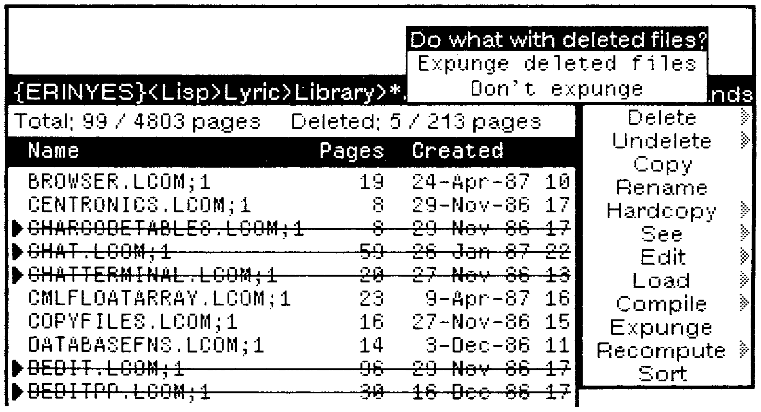
During commands of indefinite duration, such as RECOMPUTE or COPY, FileBrowser adds another command to the browser, "Abort".



Clicking on the Abort command will immediately abort the current operation. Aborting some commands can take a little while, as FileBrowser may need to do some cleaning up, so the Abort command is greyed out during this time to show you that it is doing something.

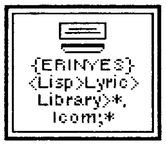
## Quitting the FileBrowser

To quit a FileBrowser, simply close the browser window. If any files have been deleted but not expunged, a small menu will pop up listing two options: "Expunge Deleted Files" and "Don't Expunge." If you choose "Expunge Deleted Files", the files will be expunged before the window closes. If you choose the "Don't Expunge" command, your deletions are ignored. If you click outside the menu, no action is taken, and the Close command is aborted.



```
Total: 99 / 4803 pages    Deleted: 5 / 213 pages
 Name                         Pages  Created
 BROWSER.LCOM;1                  19  24-Apr-87 10
 CENTRONICS.LCOM;1                8  29-Nov-86 17
▶~~CHARCODETABLES.LCOM;1           8  29-Nov-86 17~~
▶~~CHAT.LCOM;1                    59  26-Jan-87 22~~
▶~~CHATTERMINAL.LCOM;1            20  27-Nov-86 13~~
 CMLFLOATARRAY.LCOM;1            23   9-Apr-87 16
 COPYFILES.LCOM;1                16  27-Nov-86 15
 DATABASEFNS.LCOM;1              14   3-Dec-86 11
▶~~DEDIT.LCOM;1                   96  29-Nov-86 17~~
▶~~DEDITPP.LCOM;1                 30  16-Dec-86 17~~
```

If you have finished with FileBrowser only temporarily and want to put it aside to work on later, you can shrink the browser (by selecting the SHRINK command from the right-button background menu). The browser shrinks to an icon which displays the file pattern inside the browser. If any files are marked for deletion, you will be prompted with the same menu of EXPUNGE options as when you close a browser.



### Copy-Selecting Files

You can copy-select file names from a FileBrowser into other windows, such as Executive and TEdit windows, by holding down the Copy (or Shift) key while selecting a name in the window. The full name of the file is inserted as if you had typed it where the input caret is flashing. You can also copy-select in a directory-only line, in which case the full directory name is inserted in your type-in.

Note: Most file names contain characters, in particular colon and semi-colon, that have special meaning to the Lisp reader. Thus, if your type-in point is in an Executive window, you probably want to type a double-quote character before and after the file name, so that the file name is presented to Lisp as a string.

### Getting Hardcopy Directory Listings

You can get a hardcopy listing of the directory displayed in a FileBrowser by using the regular window Hardcopy command. Press the right button in FileBrowser's prompt window or tally window and select HARDCOPY from the menu. FileBrowser will produce a hardcopy listing of the files and the attributes displayed in the browser.

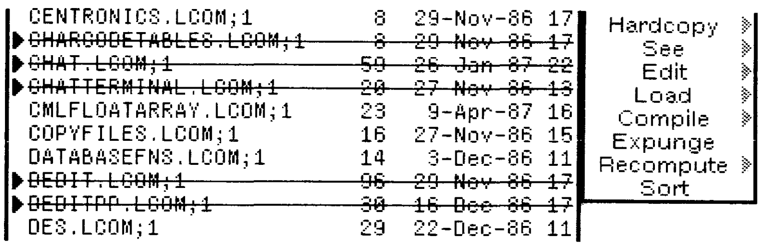
If the browser displays a large number of attributes, or your default printer font is too large, the listing may not accommodate all the attributes on one line, making the listing less readable. You may want to make the listing with fewer attributes, or use a smaller font for the listing (see description of FB.HARDCOPY.FONT in the section "Customizing FileBrowser and Using the Programmer Interface").

## FileBrowser Commands

### DELETE, UNDELETE

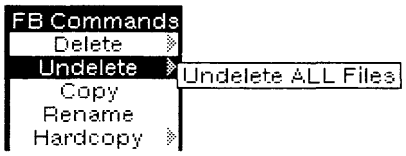
Removing a file from the file system using FileBrowser is a two-step process. First, you mark the file or files for deletion. Then you issue the Expunge command. Any time between the deletion and the expunge you can change your mind and undelete any of the files.

To mark a file or files for deletion, select them, then choose the DELETE command. FileBrowser draws a line through the deleted files. It also adjusts the numbers in the tally window to show how many files are marked deleted and how many pages they contain. It is thus easy to see how much file space you will regain when you issue the EXPUNGE command.

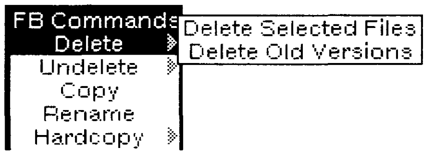


```
 CENTRONICS.LCOM;1          8  29-Nov-86 17
▶~~CHARCODETABLES.LCOM;1      8  29-Nov-86 17~~
▶~~CHAT.LCOM;1               59  26-Jan-87 22~~
▶~~CHATTERMINAL.LCOM;1       20  27-Nov-86 13~~
 CMLFLOATARRAY.LCOM;1      23   9-Apr-87 16
 COPYFILES.LCOM;1          16  27-Nov-86 15
 DATABASEFNS.LCOM;1        14   3-Dec-86 11
▶~~DEDIT.LCOM;1              96  29-Nov-86 17~~
▶~~DEDITPP.LCOM;1            30  16-Dec-86 17~~
 DES.LCOM;1                29  22-Dec-86 11
```

To undelete a file or files (i.e., to remove the deletion mark), select them, then choose the UNDELETE command. The lines through the files are removed, and the tally of deleted files is updated. The UNDELETE command has a single subcommand, UNDELETE ALL FILES, which undeletes all the files in the browser, independently of whether they are selected. This is useful if you completely change your mind about deleting any files.



The DELETE command has a useful subcommand, DELETE OLD VERSIONS. When you have been editing a file in the text editor and performing repeated PUT commands, or you the programmer have done many MAKEFILEs of the same file, multiple versions of the file accumulate, each more recent version denoted by a higher version number. The DELETE OLD VERSIONS command is used to delete excess versions of the files displayed in the browser.



To use this command, press the mouse down on the DELETE command and slide the cursor out to the right, choosing the DELETE OLD VERSIONS command. Unlike the DELETE command (or DELETE SELECTED FILES, the equivalent subcommand), the DELETE OLD VERSIONS command operates on all the files in the browser. FileBrowser prompts you for the number of versions of each file that you wish to retain. It offers the default of one version. You can accept the default or you can type a different number of your choosing, followed by a carriage return. FileBrowser then marks for deletion all but the most recent N versions of all the files in the browser, where N is the number you specified. Before issuing the EXPUNGE command, you can, if you wish, scroll through the browser, undeleting any particular files for which you wish to retain more versions than you specified.

The DELETE OLD VERSIONS command is sometimes useful even when you are not planning to actually expunge the files. This is because of the way extending the selection avoids deleted files (see the section "Selecting Files," above).

For example, if you wanted to copy only the most recent version of all the files in the browser to another location, you could do the following:

1. Use the DELETE OLD VERSIONS command, retaining just one version. This marks deleted all files but the newest version of each.

2. Go to the start of the browser and select the first file, then scroll to the end of the browser and press the right mouse button to extend the selection to the end of the browser. You have selected exactly the newest version of each file.

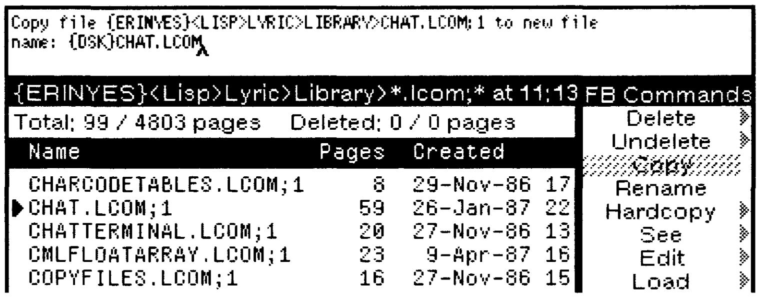
3. Use the COPY command to copy those files.

4. Finally, use the UNDELETE ALL FILES command to undelete all the old versions.

## COPY

The COPY command is used to copy an entire file or set of files to another file system location; for example, from your disk to a file server. Select the file(s) you wish to copy, then select the COPY command. FileBrowser prompts you to supply a destination.

If you selected just one file, FileBrowser prints the old name and offers a default, which consists of the same file name and either the same directory that was last used in a COPY or RENAME in this FileBrowser, or the connected directory if this is the first use of COPY in this FileBrowser. You can accept the default or supply your own destination file name. If you supply just a directory specification, e.g., {SERVER}<DIRECTORY>, the file is copied to that directory under its current name. If you supply a complete name, the file is copied to that exact name.



```
Copy file {ERINYES}<LISP>LYRIC>LIBRARY>CHAT.LCOM;1 to new file
name: {DSK}CHAT.LCOM

{ERINYES}<Lisp>Lyric>Library>*.lcom;* at 11:13
Total: 99 / 4803 pages    Deleted: 0 / 0 pages
 Name                       Pages  Created
 CHARCODETABLES.LCOM;1          8  29-Nov-86 17
▶CHAT.LCOM;1                   59  26-Jan-87 22
 CHATTERMINAL.LCOM;1           20  27-Nov-86 13
 CMLFLOATARRAY.LCOM;1          23   9-Apr-87 16
 COPYFILES.LCOM;1              16  27-Nov-86 15
```

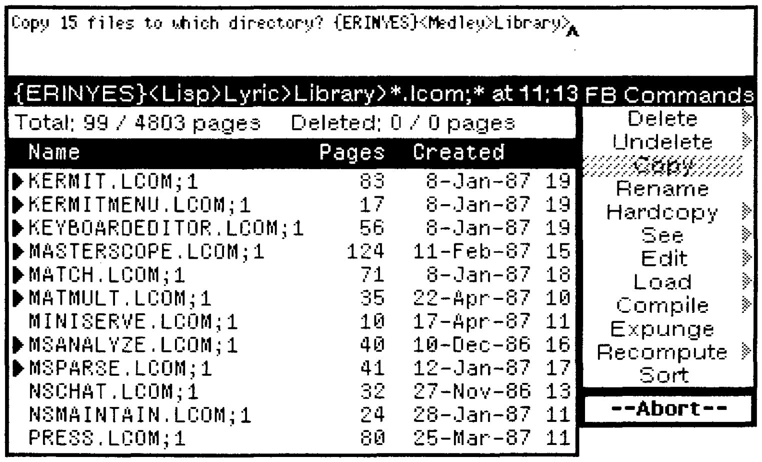
Note: Unless you specify a version number in the destination file name, the version number of the new file will be 1, or one higher than the highest existing version of the file in the destination directory, independent of the version number of the old name.

Even files marked for deletion can be copied.

If you selected several files, FileBrowser notes how many files you wish to copy and offers as a default destination the connected directory. You can accept the default or supply a different directory. All the files are copied to that directory under the names they currently have.

You must supply a directory specification, e.g., {SERVER}<YOURDIRECTORY>, rather than a complete file name, since you can't copy multiple files to the same name. If you mistakenly type a file name, rather than a directory specification, FileBrowser will complain and abort the command.

If you want to copy files from different subdirectories, FileBrowser will ask, via a message in its prompt window, if you want to preserve the subdirectory structure at the destination. If you answer YES, then the names at the destination will include not just the root name of each source file, but also all the subdirectory names below the greatest subdirectory prefix common to all the selected files (this common prefix is displayed as part of the question). If you answer NO, then the names at the destination are formed solely from the root name of each file (the name displayed in the browser), ignoring any directory information each name might have. This can cause multiple files with the same root name to be copied into the same destination name (but with different version numbers, of course).



```
Copy 15 files to which directory? {ERINYES}<Medley>Library>

{ERINYES}<Lisp>Lyric>Library>*.lcom;* at 11:13
Total: 99 / 4803 pages    Deleted: 0 / 0 pages
  Name                          Pages   Created
▶KERMIT.LCOM;1                    83     8-Jan-87 19
▶KERMITMENU.LCOM;1                17     8-Jan-87 19
▶KEYBOARDEDITOR.LCOM;1            56     8-Jan-87 19
▶MASTERSCOPE.LCOM;1              124    11-Feb-87 15
▶MATCH.LCOM;1                     71     8-Jan-87 18
▶MATMULT.LCOM;1                   35    22-Apr-87 10
 MINISERVE.LCOM;1                 10    17-Apr-87 11
▶MSANALYZE.LCOM;1                 40    10-Dec-86 16
▶MSPARSE.LCOM;1                   41    12-Jan-87 17
 NSCHAT.LCOM;1                    32    27-Nov-86 13
 NSMAINTAIN.LCOM;1                24    28-Jan-87 11
 PRESS.LCOM;1                     80    25-Mar-87 11
```

When copying (or renaming) multiple versions of the same file, FileBrowser does the copying in order of increasing version number, so that the versions at the destination are in the same relative order as at the source.

As each file is copied, FileBrowser prints a message giving the full name of the new file. If a file with the chosen name already exists, the new file's version number will be one higher; otherwise it will be version 1 (one). The new file will have the same creation date as the original file. If the destination file happens to be one that matches the pattern of the files in the browser, the new file is inserted in the appropriate place in the browser display. However, if it matches the pattern of some other FileBrowser, it is not inserted in that other browser's display (in other words, FileBrowsers do not know about each other). You would have to RECOMPUTE the destination FileBrowser to see that the file was copied into it.

## RENAME

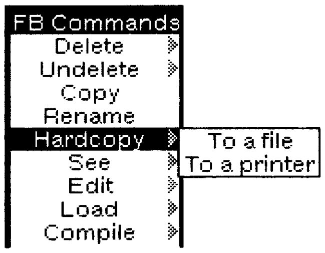
The RENAME command is used for changing the name of a file or group of files, or for moving a file or group of files to a different directory.

The RENAME command is used in exactly the same way as the COPY command. If you rename a single file, you can supply a complete new name or just a directory; if you rename several files, you must specify a directory. As each file is renamed, FileBrowser prints a message giving the file's new name and removes the file from the browser display. If the new name belongs in the same browser, it is inserted in the appropriate place. If for some reason a file could not be renamed, this is noted in the FileBrowser prompt window. The reasons for the failure of a renaming operation are roughly the same as for the failure of an EXPUNGE; the file is open, or you do not have the access rights needed to rename the file.

Note: If the destination of the rename is on a different file system than the original file, changing its name is equivalent to copying the file to its new name and then deleting the original file.

## HARDCOPY

You can print text files, TEdit files, Interpress or Press files, and Lisp files from FileBrowser. Select the appropriate file or files, then select the HARDCOPY command. The HARDCOPY command will determine what type of file you are printing and call the appropriate function for printing that file. Then the files will be printed one at a time on your default printer. The prompt window will display status messages telling you when files are being printed and when they are done (if your printer is one that provides this status service).



You may specify printing to a file or to a printer other than the default printer by means of a submenu from the HARDCOPY command. This menu is the same as the one on the HARDCOPY command in the background menu. Selecting TO A PRINTER presents you with a choice of printers from a menu. Selecting TO A FILE prompts you to supply a file name. If you selected a single file, you must specify a single hardcopy file name (or accept the offered default). If you selected multiple files, then you must specify a pattern with a single asterisk somewhere in the "name" field, for example, *.INTERPRESS or Hardcopy-*.IP. The output file names are constructed by merging the pattern with each selected file name. If the name includes an extension that implies the type of print format (e.g., .IP or .INTERPRESS implies the Interpress print format), then a file of the specified type is made automatically. Otherwise, you are prompted to supply a print format type.

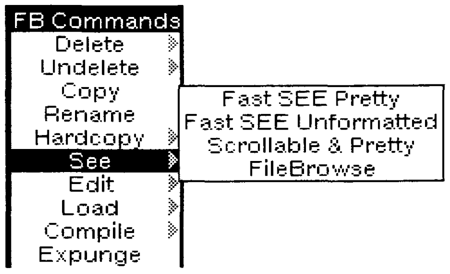
Note: For files stored on servers not supporting random access, FileBrowser is currently unable to determine that a file is in TEdit format unless the file has the extension .TEDIT. Therefore you should use TEdit to hardcopy TEdit files with other extensions. Use FileBrowser's EDIT command (to call TEdit), then the HARDCOPY command either from the TEdit Expanded Menu or from the right-button menu. As of the Lyric release, TEdit files written directly to an NS file server are identifiable as TEdit files, so this restriction does not apply to them.

Note: To obtain a hardcopy of the directory itself, use the Hardcopy command from the right-button window menu. See the section "Getting Hardcopy Directory Listings".

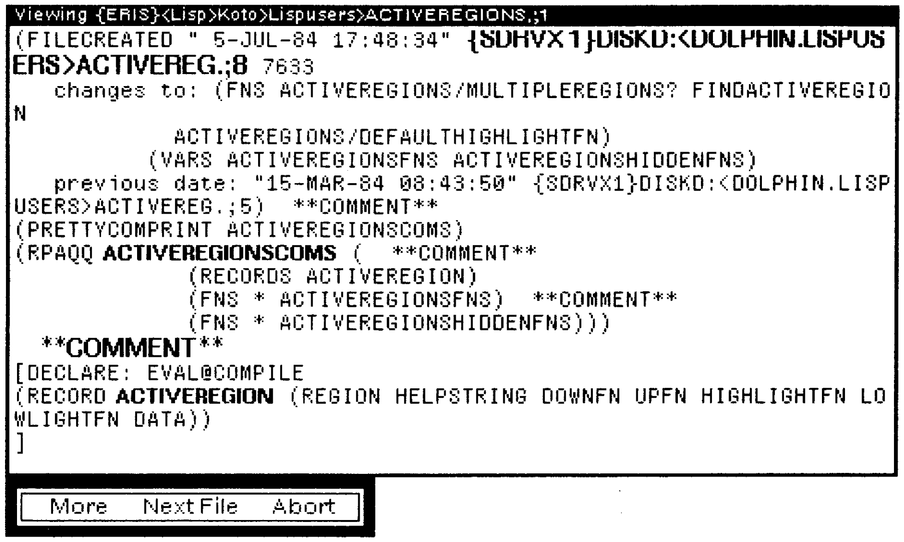
## SEE

When you browse a directory you sometimes want to see a file before printing or performing some other operation on it. To do this, select the file, then select the SEE command from the command menu. FileBrowser will prompt you to open a window by presenting you with a dashed rectangle and printing a message in the system prompt window. The window will be blank until FileBrowser starts printing the contents of the file in it.

There are actually four different SEE commands, as shown in the submenu for the SEE command. The two FAST SEE commands are provided to let you quickly see the contents of a file, but not do anything fancy, such as scroll around at random in the file. The slower SCROLLABLE & PRETTY command does let you scroll, and if the file contains formatting information of a kind that FileBrowser knows about (via the editors you have loaded), you will see the file formatted. However, this command does much more work, and may take a bit longer to show you even the first line of the file. The FILEBROWSE command is for use on "files" that are actually directories; it is described in the next section.



The two FAST SEE commands display the selected file in the display window one windowfull at a time. When the file fills the window, a small menu appears at the bottom-left corner of the window (or top-left if your display window is at the bottom of the screen) giving you the option of seeing more of the file or aborting the SEE command. If you issued the SEE command with more than one file selected, you also have the choice of aborting just the display of this file or the entire SEE command.



```
Viewing {ERIS}<Lisp>Koto>Lispusers>ACTIVEREGIONS.;1
(FILECREATED " 5-JUL-84 17:48:34" {SDHVX1}DISKD:<DOLPHIN.LISPUS
ERS>ACTIVEREG.;8 7633
   changes to: (FNS ACTIVEREGIONS/MULTIPLEREGIONS? FINDACTIVEREGIO
N
            ACTIVEREGIONS/DEFAULTHIGHLIGHTFN)
          (VARS ACTIVEREGIONSFNS ACTIVEREGIONSHIDDENFNS)
   previous date: "15-MAR-84 08:43:50" {SDRVX1}DISKD:<DOLPHIN.LISP
USERS>ACTIVEREG.;5)  **COMMENT**
(PRETTYCOMPRINT ACTIVEREGIONSCOMS)
(RPAQQ ACTIVEREGIONSCOMS (  **COMMENT**
             (RECORDS ACTIVEREGION)
             (FNS * ACTIVEREGIONSFNS)  **COMMENT**
             (FNS * ACTIVEREGIONSHIDDENFNS)))
  **COMMENT**
[DECLARE: EVAL@COMPILE
(RECORD ACTIVEREGION (REGION HELPSTRING DOWNFN UPFN HIGHLIGHTFN LO
WLIGHTFN DATA))
]
```

If you select More, the SEE command displays another windowful of the file. If you select Next File, the SEE command closes this file and goes on to display the next file in the current selection. If you select Abort, the entire SEE command is aborted. You can also abort the SEE command by closing the display window.

The next time you give a FAST SEE command, the same window will be reused.

The only difference between the FAST SEE PRETTY and the FAST SEE UNFORMATTED commands is the manner in which the characters of the file are processed as they are displayed.

The pretty (formatted) version interprets certain control characters found in Lisp source files to be font change commands, and interprets certain multibyte sequences as representing characters in the Xerox extended character set (see *XSIS Character Code Standard,* version *XC1-2-2-0*). It also squeezes out blank lines and shrinks the indentation of indented lines in order to better fit the text in a window that is generally much narrower than the standard file width. The formatted version is thus most appropriate for viewing source files and files containing plain text.

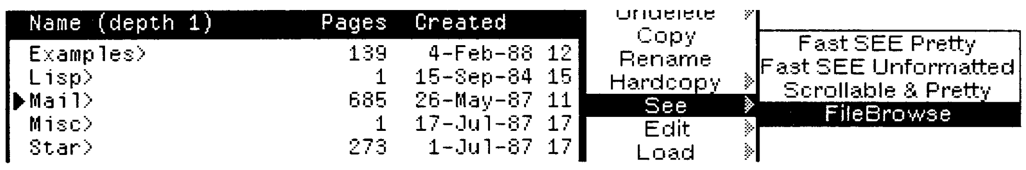
The unformatted version of the SEE command does no special processing on the characters whatsoever. It simply displays each eight-bit byte as a single character, uninterpreted. This means that bytes that do not represent normal printing characters may be displayed as black boxes, in the form ↑x or #x, or as a flashing of the window (for the byte that represents the ASCII "bell" character). The Unformatted version is thus most appropriate for viewing binary files that also contain text portions that might be worth seeing; e.g., compiled files (those with extension .MCOM) or Interpress masters (extension .IP or .INTERPRESS).

The SEE SCROLLABLE & PRETTY command views a file in a different way. This command brings up a new read-only TEdit window for viewing a file (only if TEdit is loaded in your system; otherwise, SCROLLABLE & PRETTY reverts to FAST). You can scroll and copy-select the file's contents at will, as with any TEdit window. If the file is a Lisp source file, its contents are first formatted into a TEdit document, so that all the font information is retained. This formatting, however, can take a long time for a large file. For other kinds of files, the SEE SCROLLABLE & PRETTY command is exactly like viewing the file in a regular TEdit window, except that you can't edit it. If you want to edit a file, use the Edit command instead of the See command.

You can keep the display window used by the SEE SCROLLABLE & PRETTY command open as long as you like. The command uses a different window for each file you select. Simply close the window with the standard right-button window menu when you are finished with it.

## FILEBROWSE

The FILEBROWSE command is a subcommand of SEE used to view a subdirectory in its own FileBrowser window. The selected file must be a (sub)directory. Subdirectory files appear in browsers on XNS file servers when the depth is finite (see the SET DEPTH command), and their names always end in ">".

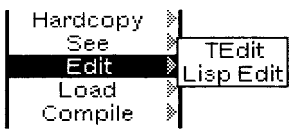


```
 Name (depth 1)          Pages  Created
 Examples>                 139    4-Feb-88 12
 Lisp>                       1   15-Sep-84 15
▶Mail>                     685   26-May-87 11
 Misc>                       1   17-Jul-87 17
 Star>                     273    1-Jul-87 17
```

On Unix servers, subdirectories are not syntactically distinguishable from ordinary files, nor is Lisp able to distinguish them internally; you simply have to know. The FILEBROWSE command prompts you for a region for a new FileBrowser window group, in which it proceeds to enumerate the contents of the selected subdirectory, to the same depth as the main browser used, if any.

## EDIT

The EDIT command invokes an editor on the selected file. To specify an editor explicitly, use one of the commands on the submenu.



To start up a TEdit editor on a selected text file, select EDIT with the left mouse button. If you have recently closed a TEdit window, then TEdit will probably reuse that window; otherwise, you will be prompted to create an editor window. TEdit only remembers the most recently abandoned window, however, so if you issue the EDIT command when you have several files selected, you will be prompted to create windows for all but the first file.

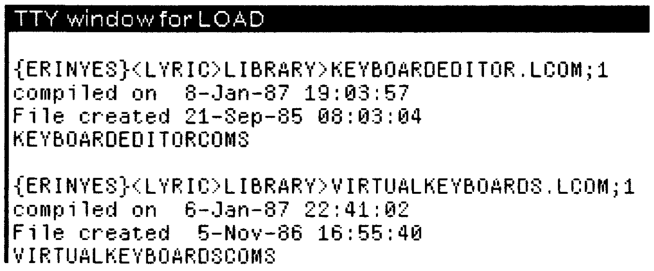
The subcommand LISP EDIT is appropriate for Lisp source files produced by the file manager. It calls the Lisp structure editor on the file's coms. If the file is not yet known to the file manager, you will be asked whether you want to load it first (using LOAD PROP). If not, the operation is aborted. The editor used is the default structure editor (SEdit or DEdit, depending on your setting of *EDITMODE*).

If you select the main Edit command, without sliding off to the submenu, then FileBrowser's default editor is called. This editor is initially TEdit, but you can change the default behavior by setting the variable FB.DEFAULT.EDITOR (see the section "Customizing FileBrowser and Using the Programmer Interface," below).

## LOAD

FileBrowser's LOAD command can be used to load both source (interpreted) and compiled files into your workstation's virtual memory. First select the file or files you want to load, then select Load with the left mouse button.

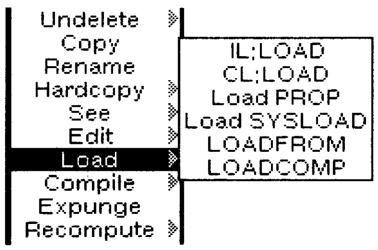
A special display window is opened to give information about the files as they are loaded. When the load is complete, FileBrowser closes the load window.


TTY window for LOAD

```
{ERINYES}<LYRIC>LIBRARY>KEYBOARDEDITOR.LCOM;1
compiled on  8-Jan-87 19:03:57
File created 21-Sep-85 08:03:04
KEYBOARDEDITORCOMS

{ERINYES}<LYRIC>LIBRARY>VIRTUALKEYBOARDS.LCOM;1
compiled on  6-Jan-87 22:41:02
File created  5-Nov-86 16:55:40
VIRTUALKEYBOARDSCOMS
```

The LOAD command also has subcommands that enable you to load files in different ways. These commands are described briefly here; see the *IRM* for further details. Only the LOAD and LOAD SYSLOAD commands are of interest to nonprogrammers. All load commands are placed on the history list. With the exception of LOAD SYSLOAD, all are undoable using Lisp history list commands.



LOAD (same as the Lisp LOAD command) loads the file with LDFLG = NIL. If any functions or variables in the file redefine ones that are already in memory, messages such as "(FOO redefined)" are printed. This command is used for loading source files that you as a programmer plan to make modifications to, or for loading any file that you have doubts about, in which case you might want to be able to undo the load.

CL:LOAD differs from IL:LOAD in that it calls the Common Lisp LOAD function, rather than that of Interlisp. In the present implementation, these two commands are essentially identical.

LOAD PROP loads the file(s) with LDFLG = PROP. Interlisp functions are loaded onto property lists, rather than redefining the functions already in memory. Common Lisp functions, macros, etc. are loaded into a definitions table, again without changing the definitions currently in effect. This command is used for loading Lisp source files for which the compiled version already exists in memory, but which you plan to edit.

LOAD SYSLOAD loads the file(s) with LDFLG = SYSLOAD. Function and variable redefinitions occur quietly (i.e., without printing (FOO redefined), (BAR reset), etc.). The file manager is not informed of this file. This is the fastest loading command and consumes the fewest resources, but it is not undoable. It is the best way to load compiled (extensions LCOM or DFASL) files that you are certain you want to load into your environment and are not planning to edit.

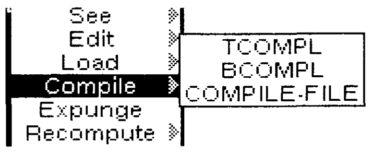
LOADFROM calls the file manager's function LOADFROM. This loads variables and other expressions but not Interlisp functions, and does so in a way that informs the file manager, so that the editor knows where to find the functions.

Note: The LOADFROM command is not appropriate for Common Lisp files—it is better to use LOAD PROP for them.

The LOADCOMP subcommand calls the file manager's function LOADCOMP. This loads from the file all the expressions that the compiler would evaluate if compiling the file—macros, records, and any other expressions enclosed in an EVAL@COMPILE declaration.

## COMPILE

The COMPILE command is used to compile a selected Lisp source file or files. The files do not have to be loaded. The COMPILE command uses the same compiler as CLEANUP does (the value of *DEFAULT-CLEANUP-COMPILER*), unless you select a different compiler from the COMPILE submenu.



Note that this command is placed on the history list, so that it and its subcommands are undoable.

A special Executive window is opened for each file to display information about the code being compiled. When the compilation is finished, the window is closed. In the case of

TCOMPL and BCOMPL, which invoke the Interlisp compiler, each compiled file is saved on your connected directory with the original file name and the extension MCOM. In the case of COMPILE-FILE, which invokes the Xerox Common Lisp compiler, the compiled file is saved on the same directory as the source file with the original file name and the extension MFASL.

Note that this command compiles files found on a storage device, not the functions defined in the Lisp image. If you have made changes to any of the functions on a loaded file, you must perform a MAKEFILE to write an updated version before compiling it. For more information on making files and compiling, see the *IRM*.

## EXPUNGE

If you are sure you want to delete files permanently, choose the EXPUNGE command. The EXPUNGE command is grayed while FileBrowser expunges the files that were marked for deletion by the DELETE command. As each file is removed from the system, it is removed as well from the browser display, and the tally of total number of files and number of deleted files is updated, so you can see the progress of the command.

If for some reason a file can not be expunged, FileBrowser prints a message saying so in its prompt window, but continues to expunge the other files. The main reasons that prevent a file from being expunged are its being opened, either by you or some other user, or your not having the access rights required to delete it (if it is on a file server). See the section "Troubleshooting Problems with FileBrowser," below.

Note: The EXPUNGE command is not affected by the current selection; it operates only on files marked for deletion, whether currently selected or not.

## RECOMPUTE

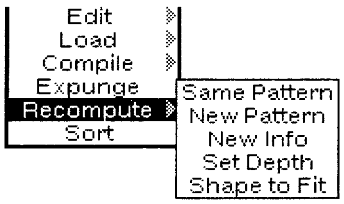
FileBrowser's display shows those files that existed and matched the specified pattern at the time you created the browser. If you want the browser to reflect the latest state of the file system, use the RECOMPUTE command.

For example, if you open a FileBrowser on your directory, then save several versions of a TEdit file on that directory, the file listing will not display the new versions until you RECOMPUTE.

The RECOMPUTE command operates exactly as when you started up FileBrowser initially: it clears the display and tally windows, then enumerates the files matching the pattern. The RECOMPUTE command in the menu is grayed until the enumeration is finished. During this time you cannot scroll or perform any other operations on the browser. However, you can close the window if you want to abort the command and throw away the browser.

If any files are marked for deletion at the time you request a RECOMPUTE, FileBrowser will present the choice of expunging or undeleting the files, just as it does when you want to quit the browser (see the section "Quitting the FileBrowser," above).

The RECOMPUTE command also has a menu of subcommands that allow you to list different files or different information for the same set of files.



SAME PATTERN is the same as the main RECOMPUTE command, i.e., it enumerates the files matching the same pattern as before.

NEW PATTERN lets you change the pattern, i.e., browse a new set of files. FileBrowser prompts you to supply a new file name pattern and offers the old pattern as an initial default. You can either type an entirely new pattern, replacing the one offered, or delete the old pattern one character at a time by backspacing. Press the carriage return when you have finished specifying the pattern. FileBrowser then enumerates the files matching this pattern, just as with the RECOMPUTE command. You can abort the command with the Abort button, or by erasing the whole pattern (by backspacing or using control-Q) and then pressing the carriage return.

NEW INFO lets you change which attributes the browser displays. It is described in the next section.

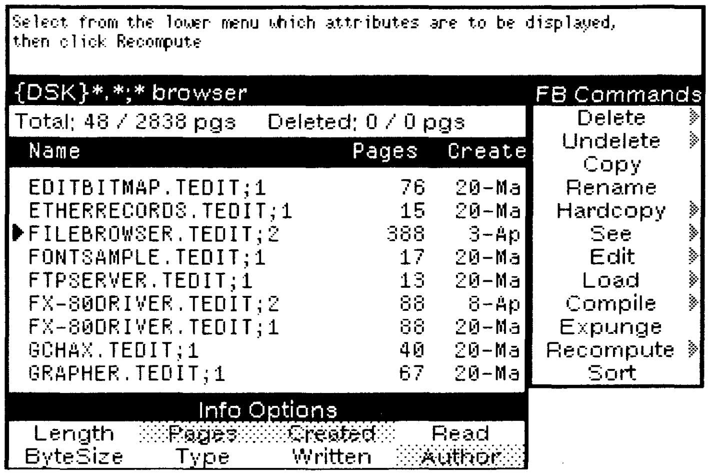
SET DEPTH lets you change the depth to which FileBrowser enumerates a directory on an XNS file server. It is described in the section "SET DEPTH".

SHAPE TO FIT reshapes the FileBrowser window so that all the attributes in the display are visible at once, eliminating the need to horizontally scroll the window to get at all the information.

## NEW INFO

FileBrowser displays some file attributes, or information about the file, alongside each file in the browser display. Ordinarily, the attributes displayed are the size of the file in pages, its creation date, and its author. You can change which attributes are displayed for all new FileBrowsers by changing FB.DEFAULT.INFO (see the section "Customizing FileBrowser and Using the Programmer Interface," below). You can change the attributes displayed in a particular browser window by using the NEW INFO command.

To use the NEW INFO command, select it from the submenu of the RECOMPUTE command. FileBrowser opens up an additional subwindow, the Info Options window, below the display window. This subwindow contains a menu of attributes, with the current defaults shaded. Selecting a shaded item unshades it; selecting an unshaded item shades it. When you have selected all the attributes you wish to see displayed, issue either the RECOMPUTE or the NEW PATTERN command. The files will be listed with the new information you requested. The Info Options window stays open—you can close it at any time.



```
Select from the lower menu which attributes are to be displayed,
then click Recompute

{DSK}*.*;* browser
Total: 48 / 2838 pgs    Deleted: 0 / 0 pgs
 Name                          Pages  Create
 EDITBITMAP.TEDIT;1               76  20-Ma
 ETHERRECORDS.TEDIT;1             15  20-Ma
▶FILEBROWSER.TEDIT;2             388   3-Ap
 FONTSAMPLE.TEDIT;1               17  20-Ma
 FTPSERVER.TEDIT;1                13  20-Ma
 FX-80DRIVER.TEDIT;2              88   8-Ap
 FX-80DRIVER.TEDIT;1              88  20-Ma
 GCHAX.TEDIT;1                    40  20-Ma
 GRAPHER.TEDIT;1                  67  20-Ma
```

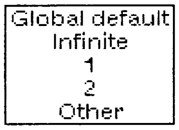
The Info Options items have the following meanings:

| | |
|---|---|
| CREATED | The date and time that the content of the file was created. This date changes whenever the file is modified, but does not change when a file is copied or renamed. |
| WRITTEN | The date and time the file was last written to the file system. This date is never older than the Created date, but it can be newer if the file is copied, unmodified, from one file system to another. |
| READ | The date and time the file was last read. This attribute may be blank if the file has never been read. |
| AUTHOR | The login name of the person who wrote the file, or last modified it. |
| LENGTH | The length of the file in (usually eight-bit) bytes. |
| PAGES | The number of 512-byte pages in the file. On some servers, this attribute is blank if the file is empty. |
| BYTESIZE | The size (in bits) of the bytes in the file. In Xerox Lisp this is always eight, but some older computers and file servers allow other sizes. |
| TYPE | A value indicating what kind of data the file contains. The usual values of this attribute are TEXT, meaning the file contains just characters, or BINARY, meaning the file contains arbitrary data. Some servers have additional types, such as INTERPRESS for files in Interpress format. |

## SET DEPTH

XNS file servers support a feature that allows enumerating a directory to a user-specifiable depth. The "depth" of a file reflects the number of subdirectories between it and the root of the enumeration, i.e., the directory or subdirectory you gave in the pattern to FileBrowser, not counting any containing wildcards (asterisks). The immediate descendants of the root are at depth 1, files in subdirectories of depth 1 are at depth 2, and so on.

Ordinarily, FileBrowser enumerates a directory to the default depth, which is usually unlimited. To enumerate a directory to a different default, use the FB command with argument :DEPTH *n*, for some positive integer *n*, or T for unlimited depth. To change the depth in an existing FileBrowser, use the SET DEPTH command, a subcommand under the RECOMPUTE command. The command offers you a menu of choices:



"Global default" means use the default depth, overriding the depth at which this browser was last enumerated. "Infinite" means use no depth limit (same as depth T). "1" and "2" are common depth choices; to choose some other numeric value, select "Other" and enter the value via the displayed keypad.

The SET DEPTH command does not affect the current display. It takes effect the next time you use the RECOMPUTE or FILEBROWSE commands from the same browser.

During a RECOMPUTE, if a subdirectory appears at the specified maximum depth, its descendants are not enumerated; rather, the subdirectory itself appears as an entry in the browser display. This entry can be selected, just like a file, but only a small number of commands can be used on it: you can RENAME it, you can DELETE it if it has no descendants, and you can FILEBROWSE it. It has attributes, just as ordinary files do. Its page size is the size of the entire subtree rooted at the subdirectory.

Note: Depth currently affects only XNS servers; all other devices ignore it and enumerate to their own default depths. In addition, due to a bug in XNS Services 10, depth is ignored for nontrivial patterns, i.e., anything but "*.*".

## SORT

The SORT command allows you to sort the files in the browser by any attribute of the files displayed in the browser. Selecting SORT brings up a menu of attributes by which to sort. This menu includes all the attributes currently displayed in the browser (such as Creation Date, Author), plus the choice Name. For some attributes you can sort forward or backwards; the choice is on a submenu, and the default is generally in the order of numerically greatest (e.g., size) or most recent (e.g., creation date) first.

If the attribute you select is not Name, then the file names displayed in the browser will be reformatted to include their directory portion (if there are any subdirectories below the browser's main pattern), as the subdirectory information is no longer implicit in a file's position in the browser.

The sort order Name, Decreasing Version is the default order in which browsers initially are created.

## Customizing FileBrowser and Using the Programmer Interface

FileBrowsers are created programmatically by the function FILEBROWSER, or by type-in from an Executive window with the FB command.

Note: All functions, variables and literals in this section are in the Interlisp package. You'll have to use the prefix IL: if you are using another Executive.

### FileBrowser Functions

(FILEBROWSER *FILESPEC ATTRIBUTES OPTIONS*) [Function]

Creates a FileBrowser on files matching the pattern *FILESPEC* and displaying the values of the specified *ATTRIBUTES* for each file. Returns the main window of the browser.

If *ATTRIBUTES* is missing or NIL, it defaults to the value of FB.DEFAULT.INFO (below). See FB.INFO.MENU.ITEMS for the complete set of allowable attributes.

*OPTIONS* is a list in property-list format. The currently implemented properties and their values are as follows:

| | |
|---|---|
| :REGION | A screen region in which FILEBROWSER will open the browser; if this option is omitted, FileBrowser will prompt you for a region. |
| :DEPTH | The depth to which the enumeration should be performed. Affects only XNS servers (see SET DEPTH). |
| :TITLE | The title for the main browser. If this is omitted, the title derives from *FILESPEC*, and is updated whenever the RECOMPUTE command is used. |
| :MENU-TITLE | The title for the command menu. Defaults to "FB Commands". |
| :MENU-ITEMS | The set of ITEMS composing the command menu. The default is the value of FB.MENU.ITEMS. |

FB *FILESPEC ATTR1 . . . ATTRN* [Command]

This is an Executive command for creating FileBrowsers. FB creates a FileBrowser on files matching the pattern *FILESPEC* and displaying the attributes *ATTR1* through *ATTRN*, or the value of FB.DEFAULT.INFO if no attributes are specified. It prompts you for a window region. If a keyword appears in the command line, the remainder of the line from that point on is interpreted as the *OPTIONS* argument to FILEBROWSER.

For example, the Executive command

```
FB *.MCOM LENGTH CREATIONDATE
```

browses all files on the connected directory with extension MCOM, displaying the length in bytes and creation date for each. The command

```
FB * :DEPTH 1
```

browses the connected directory to depth 1, displaying the default attributes.

FB always returns NIL.

## FileBrowser Variables

There are several global variables that can be altered to affect FileBrowser's behavior. You can set them by typing

(SETQ *VARIABLENAME NEWVALUE*)

to your Executive window, and can save their values with a VARS command in your initialization file.

FB.DEFAULT.INFO [Variable]

A list specifying which attributes should be displayed for each file. The elements of this list are the Lisp names for the attributes you want displayed. The choices are CREATIONDATE, WRITEDATE, READDATE, LENGTH, SIZE, BYTESIZE, AUTHOR, and TYPE. The attribute SIZE corresponds to the info item "Pages".

For example,

```
(SETQ FB.DEFAULT.INFO '(CREATIONDATE LENGTH))
```

would cause all new FileBrowsers to display exactly the attributes creation date and length in bytes for each file.

FB.INFO.MENU.ITEMS [Variable]

The list of items in the menu used by the NEW INFO command. Each element of the list is of the form (*LABEL ATTRIBUTE* "*DOCUMENTATION*"). If you add new attributes to this variable, you should also add corresponding entries to FB.INFO.FIELDS.

FB.INFO.FIELDS [Variable]

A list describing, for each attribute, the format in which it is displayed. In addition, the order of attributes in this list is the order in which they are displayed in a browser window. Each element is of the form (*ATTRIBUTE HEADER WIDTH FORMAT PROTOTYPE*), where

*ATTRIBUTE* is the name of the attribute, which must be a valid attribute for GETFILEINFO;

*HEADER* is a string displayed in the header line above the main browser window;

*WIDTH* is the total width in pixels to allocate for printing the values of this attribute, including trailing space;

*FORMAT* is NIL for ordinary values, DATE for attributes whose value is a date, or (FIX *N*) for integer values;

*PROTOTYPE* is a string describing the widest value you expect the field to have.

All values are printed left-justified in the allotted space, except for format (FIX *N*), used for integer values, which are right-justified in a field *N* pixels wide, with *WIDTH–N* pixels left for trailing space.

If *PROTOTYPE* is present, then the *WIDTH* and *N* fields for the item are ignored, and the width is taken to be the width of the prototype string in the browser's font (FB.BROWSERFONT), plus two characters' worth of space between columns.

FB.DEFAULT.NAME.WIDTH [Variable]

The amount of space, in points, to use for displaying file names in a browser, initially 140. The name column is automatically expanded if enough names are too wide. You can set this larger if you routinely browse directories of long file names.

FB.ICONFONT [Variable]

The font in which the file pattern is displayed on the browser icon, initially eight-point Helvetica. The value of this variable should be a font descriptor, as returned by FONTCREATE.

For example,

```
(SETQ FB.ICONFONT (FONTCREATE 'MODERN 10 'BOLD))
```

FB.BROWSERFONT [Variable]

The font in which the information in the main display window is printed, initially 10-point Gacha.

FB.PROMPTFONT [Variable]

The font in which prompt messages are printed, initially eight-point Gacha.

FB.PROMPTLINES [Variable]

The number of lines in the prompt window, initially three.

FB.HARDCOPY.FONT [Variable]

Specifies the font to use when producing hardcopy directory listings. Initially NIL, which means to use the font class DEFAULTFONT.

FB.HARDCOPY.DIRECTORY.FONT [Variable]

Specifies the font to use for subdirectory names, if there are any, in hardcopy directory listings. Initially NIL, which means to use the font class ITALICFONT.

FB.DEFAULT.EDITOR [Variable]

Specifies the editor to call by default when the main Edit command is selected. Its value is one of the following:

| | |
|---|---|
| TEDIT | Use the TEdit text editor. |
| LISP | Use the Lisp structure editor, as in the Lisp Edit subcommand. |
| NIL | Use the Lisp structure editor if the selected file is a File Manager Lisp source file, TEdit otherwise. |
| Other | Names the entrypoint function of the editor of choice. The editor is called with a single argument, the file name. |

The initial value of FB.DEFAULT.EDITOR is TEDIT.

FILING.ENUMERATION.DEPTH [Variable]

The system variable that controls the default depth to which a directory is enumerated. The value is a positive integer, or T for unlimited depth. The initial value is T.

## Adding FileBrowser Commands

You can add your own commands to FileBrowser by adding items to FB.MENU.ITEMS and writing functions to handle the commands. This section describes the format of menu commands and a set of functions that are useful for the implementation of FileBrowser commands.

FB.MENU.ITEMS [Variable]

A list of the items that appear on FileBrowser's command menu. You can add new FileBrowser commands by adding new elements to the end of this list. After your change, any new FileBrowser will have the added commands.

Each element of FB.MENU.ITEMS is of the form (*LABEL YOURFN "EXPLANATION"*), where

> *LABEL* is the name of the command, as it is to appear in the menu;
>
> *YOURFN* is the name of the function to be called when the command is selected, and
>
> *EXPLANATION* is the explanation to be printed when the mouse cursor is held over the command.

While *YOURFN* is Executing, the menu command is grayed out, and FileBrowser is "locked" so that no other commands or processes can access it.

You can have subcommands as well if you make the menu command be of the form (*LABEL YOURFN "EXPLANATION"* (*SUBITEMS item1. . . itemN*)), where each *itemJ* is recursively of the same form as a menu command.

FileBrowser calls *YOURFN* with four arguments: (*YOURFN BROWSER KEY ITEM MENU*), as follows:

> *BROWSER* is FileBrowser object in control of this browser window.
>
> *KEY* is the mouse key pressed (left or middle).

*ITEM* is the menu item that was selected.

*MENU* is the command menu.

*YOURFN* can also be a list of two elements, (*FN ARG*), where *ARG* is an arbitrary value that is passed, unevaluated, as the fifth argument to *FN*. This is useful for writing one function that implements several subcommands that are similar to the original command.

Any additions to FB.MENU.ITEMS should be saved with the file manager command APPENDVARS, so that the new items are added to the end of the menu, and your changes will not interfere with any changes to built-in FileBrowser commands in new FileBrowser releases.

(FB.TABLEBROWSER *BROWSER*) [Function]

Returns the TABLEBROWSER object belonging to FileBrowser *BROWSER*. See the documentation for the module TABLEBROWSER for further operations you might perform on one of these browsers.

(FB.SELECTEDFILES *BROWSER NOERRORFLG*) [Function]

Returns a list of table items representing the files currently selected in *BROWSER*. If there are no selected files, this prints

**No files are selected**

in the prompt window, unless *NOERRORFLG* is true, in which case this function quietly returns NIL.

(FB.FETCHFILENAME *ITEM*) [Function]

Returns the full name of the file denoted by *ITEM*, one of the table items returned by FB.SELECTEDFILES.

(FB.PROMPTWPRINT *BROWSER X1...XN*) [Function]

Prints the strings *X1* through *XN* in *BROWSER*'s prompt window. The item T is printed as a carriage return (i.e., a command to go to a new line).

(FB.PROMPTW.FORMAT *BROWSER FORMAT-STRING* &REST *ARGS*) [Function]

Prints to *BROWSER*'s prompt window by applying the Common Lisp function FORMAT to *FORMAT-STRING* and *ARGS*.

(FB.PROMPTFORINPUT *PROMPT DEFAULT BROWSER ABORTFLG DONTCLEAR*) [Function]

Prompts for your input in *BROWSER*'s prompt window. *PROMPT* is the prompt string, *DEFAULT* is the default answer. Returns your input as a string, or NIL if there is no input, or if it was aborted with control-E. If there is no input and *ABORTFLG* is true, prints "...aborted". The prompt window is first cleared (as at the beginning of a command), unless *DONTCLEAR* is true.

(FB.ALLOW.ABORT *BROWSER*) [Function]

Enables the Abort button on *BROWSER*. This should be called from the function implementing any FileBrowser command of indefinite duration.

## Troubleshooting Problems with FileBrowser

When FileBrowser returns the message "No files in group *FILENAMEPATTERN*" when you know those files exist, the file server is probably down or rejecting connections. If this is so, your only option is to wait until the server is functioning again, and then give the Recompute command. In the case of an NS file server, the enumeration of files can also fail if you do not have sufficient access privileges; this condition is usually noted by a message in the system prompt window.

When you try to expunge a file and FileBrowser displays the message "Can't expunge FILENAME," it may be because you don't have write access to the file, or someone else is reading the file. However, the most common reason is that the file is still open. Be sure to close any TEdit windows in which you may still be viewing the file. If you have recently issued a HARDCOPY command for the file, a background process may still be working on the file. If that's not the problem, you can get a list of open streams by typing (OPENP) at the prompt in an Interlisp Executive window. If one of them is open on the file you want to delete, you can close it by passing the stream to the function CLOSEF. To close all open streams (not recommended unless you're sure it is okay), you can type (MAPCAR (OPENP) 'CLOSEF). If you don't find an open stream, and the server is a Leaf or XNS file server, you may have a disagreement between Lisp and the server on what files are open, something that can occur if you had aborted an OPENSTREAM operation. Call (BREAKCONNECTION "servername") to reset the connection, then try again.

[This page intentionally left blank]

# FILEWATCH

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

May 14, 1987

## SUMMARY

FILEWATCH is a facility for keeping an eye on open files. It continually updates a display showing each open file, its current file pointer location, the total file size, and a percentage bar.

## 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 (user beware!), 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, file cacher files). Initially set to T if you are running the Koto release of Interlisp-D, otherwise NIL. (Later releases have no facility yet to obtain all open files.)

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).

# FONTSAMPLE

FontSample provides a set of tools for generating Interpress masters for a font sampler, and is intended to allow you to see what results on a printer when you specify a font. This is useful because there may be several font substitutions between when you specify a font (for example in TEdit) and when a printer actually renders the font.

The set of font mappings is a function of the local font substitutions in a particular workstation, the workstation's environment (which fonts are available to it at that time, if the font file server is temporarily unavailable), which printer is being used, and which font files are currently installed on the printer.

For example, Lisp might substitute Terminal for Gacha, and the printer might substitute Modern for Terminal. Thus you could request Gacha and get Modern. The font sampler is intended to display the final result of these substitutions.

## Requirements

FONTSHEETSx.IP (where x = 1, ... 7)

## Installation

Load FONTSAMPLE.LCOM from the library.

## User Interface

To find the cumulative effect of all font substitions at a given time, environment, and printer, you should generate the Interpress masters (which reflect the environmental and Lisp mappings) and then send them to various printers (which reflect the printer mappings).

```
(SEND.FILE.TO.PRINTER 'FONTSHEET1.IP)
(SEND.FILE.TO.PRINTER 'FONTSHEET2.IP)
etc.
```

Short sample masters for all currently supported Interpress fonts can be found in the Lisp Library with the names FONTSHEET1.IP, FONTSHEET2.IP ... FONTSHEET7.IP. These reflect the default Lisp font mappings.

Note: These environmental mappings may change from release to release, so new masters should be printed for every release.

## Functions

(FNT.MAKEBOOK *OUTFROOTNAME LISTOFFONTS PRINTFN PERPAGE)*

[Function]

This function enumerates fonts across multiple output files. Multiple files are necessary because some printers cannot handle documents with many fonts. The function iterates over the fonts in *LISTOFFONTS*, invoking *PRINTFN* (which has arguments as in FNT.DISPLOOK) *PERPAGE* times per output file. The files are named *OUTFROOTNAME* with the page number concatenated at the end. If *LISTOFFONTS* is the atom ALL, then FONTSAVAILABLE is invoked for all Interpress fonts (this is an extremely slow operation). *PRINTFN* defaults to FNT.DISPLOOK; *PERPAGE* defaults to 18. Each font in *LISTOFFONTS* should be a FONTLIST.

(FNT.DISPTBLE *STREAM FONT*) [Function]

This function prints a description of the font and characters 0 to 254 of the font specified by *FONT* on *STREAM.* It expects a letter-sized output stream. If used with FNT.MAKEBOOK, *PERPAGE* should be 1.

(FNT.DISPLOOK *STREAM FONT*) [Function]

This function prints a description of the font and representative characters of *FONT* on *STREAM*. If used with FNT.MAKEBOOK, *PERPAGE* should be 18.

## Limitations

The font sheets are applicable only to Interpress printers.

## Example

```
(SETQ FONTLISTLIST (LIST '(MODERN 10 MRR 0 INTERPRESS)
'(CLASSIC  8 BRR 0 INTERPRESS)))
```

```
(FNT.MAKEBOOK 'FOO FONTLISTLIST 'FNT.DISPTBLE 1)
```

generates two files named FOO1 and FOO2 each containing output from one invocation of FNT.DISPTBLE. Thus FOO1 has a font table for (MODERN 10 MRR 0 INTERPRESS), FOO2 has a font table for (CLASSIC 8 BRR 0 INTERPRESS).

# FTPSERVER

FTPServer implements a simple PUP FTP server protocol for a Xerox workstation. The server is typically run as a background process on one machine to allow other machines remote access to the files on its disk.

## Requirements

Ethernet connection to a remote host.

## Installation

Load FTPSERVER .LCOM from the library.

## Functions

(FTPSERVER *FTPDEBUGLOG*) [Function]

Creates a process named FTPSERVER that listens on the standard PUPFTP server socket for incoming connection requests. When one arrives, FTPSERVER services it, then returns to its listening state. The process continues to run until killed.

If *FTPDEBUGLOG* is non-NIL, it should be an open file/stream to which tracing information is printed during the life of the process.

If *FTPDEBUGLOG* is T, output goes to a newly created window. *FTPDEBUGLOG* can also be a REGION, specifying where the window is to be created.

FTPSERVER.DEFAULT.HOST [Variable]

Initially DSK. This is the default host for files requested of the server via FTP. Setting this to FLOPPY, for example, would serve files off the machine's floppy drive.

Note: FTPSERVER.DEFAULT.HOST can also be set to the name of a remote host, but this has limited utility, as it doesn't handle passwords correctly.

## Limitations

The current implementation is a simple tool which allows file transfer between Xerox machines and supports only one remote connection at a time. Because of this, files cannot be loaded indirectly, i.e., via the filecoms of another file.

For example, suppose FOO loads BAR which loads WOO. When FOO is being loaded, it will attempt to load BAR. But FTPServer cannot support the second connection required to load BAR while the first connection is still open to load FOO. (This is similar to the case of trying to load FOO and BAR when they are on different floppies.)

Therefore, you should load files in an order that prevents recursive loads: in this example, load WOO, then BAR, then FOO.

Delete (DELFILE) operation is now supported. Rename (RENAMEFILE) operation is not implemented. FTPServer is best suited for simple COPYFILE operations.

## Examples

An alternative way of specifying the host from the remote machine is to make the host name be the "device" field of the file name specification.

For example, if machine M is running FTPServer, another machine could ask for directory of {M}FLOPPY:FOO.* to get a listing of M's {FLOPPY}FOO*.

To address your host, you may use the results of ETHERHOSTNAME. If on your host (ETHERHOSTNAME NIL T) evaluates to 123#456#, then on a remote machine you can access file FOO on the host by:

```
{123#456#}FOO
```

# FX-80DRIVER

FX-80Driver prints text and graphics on Epson FX-80-compatible printers. It implements a full device-independent graphics interface for the FX-80, and can print source code, TEdit documents, bitmaps and windows at a variety of qualities and speeds.

The FX-80Driver contains two printer drivers: a fast driver, for quick printing of draft-quality text, and a high-quality driver, for slower printing of mixed-font text and graphics. You can print early revisions of a document in fast mode, and then switch to high-quality mode for the final copy. The matrix shown in Figure 1 illustrates the capabilities of each mode:

| | Fast | High-quality |
|---|---|---|
| TEdit | monofont only | yes |
| Sketch | | yes |
| Windows | | yes |
| Lisp source code | monofont only | yes |
| Grapher | | yes |

*Figure 1. FX-80 printer drivers*

For historical reasons, FX-80 in this document refers to any and all of the Epson FX-80 family of dot-matrix graphics printers. The module supports the FX-80, FX-85, FX-86 and FX-286. The Epson printers vary in speed and carriage width, but share a common command language.

## Requirements

RS232 or TTY cable (see the wiring diagrams in the Introduction of this manual).

Serial interface card in the printer.

DLRS23C or DLTTY.

## Installation

### FX-80 Serial Interface

The FX-80Driver module requires that your Epson be equipped with a suitable serial interface (such as the Hanzon Universal Card).

The interface should be set up with XOn/XOff flow control enabled, 9600 baud or slower, 1 stop bit, 8 bit characters, no parity.

(See *The Hanzon Universal Card* booklet for instructions on the DIP switch settings.)

### FX-80 DIP Switch Settings

The FX-80 should have its DIP switches set as shown in Figure 2.

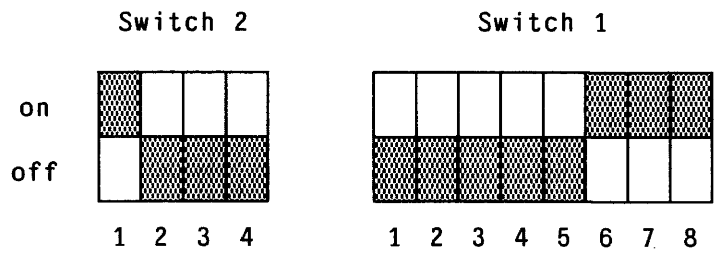


*Figure 2. FX-80 DIP switch settings*

Switch 1 says no automatic linefeed, no automatic paper feed, no buzz on paper-out, and to allow no software deactivation of the printer.

Switch 2 says to use the USA character set, Pica type, allocate 2KB for user-defined characters, allow paper-out detection, and print zeros as zeros.

Note: For the FX-85, -86 and -286 DIP switch settings, consult the corresponding *Epson User's Manual*.

### Software

Load FX-80DRIVER.LCOM and the required .LCOM modules from the library.

Store all of the font files (file names ending with .displayfont) corresponding to the fonts you wish to use on some convenient directory or directories. HQFX80-FONT-DIRECTORIES should be a list that contains these directories; it should be the same as DISPLAYFONTDIRECTORIES.

Set FASTFX80-DEFAULT-DESTINATION (determines where output to the FASTFX80 lineprinter device goes) and HQFX80-DEFAULT-DESTINATION (determines where output to the HQFX80 lineprinter device goes) to one of the following values; they need not be the same:

| Destination | RS232 port | TTY port | file |
|---|---|---|---|
| Value | {RS232} | {TTY} | FileName |
| Speed | 9600 max. | 4800 max. | n/a |

Load the appropriate device driver for each of these destinations: DLTTY.LCOM for the TTY port, and DLRS232C.LCOM for the RS232C port.

Run the function RS232C.INIT or TTY.INIT (as appropriate), and set the baud rate to match the setting on the printer.

## User Interface

You can set up the FX-80 to be your default printer, send FX-80 output to a file for later printing, or programmatically open an image stream that produces output on the FX-80.

Having the FX-80 set up as your default printer means that you can print the contents of windows by selecting the HARDCOPY menu item on the window of interest. You can also use the HARDCOPY - TO A FILE submenu item to spool your output for later printing. And you can write programs that use the OPENIMAGESTREAM function to create FX-80 format graphics output.

## Printing in Fast Mode

You can print in fast mode by sending output to the printer FASTFX80 or by opening an image stream to a file with extension FASTFX80. This mode is called fast because it uses the printer's built-in font, which allows a tight encoding of the document to be printed. Fidelity to the original document is not as good as in high-quality mode.

The following restrictions apply:

> Special characters (that is, most Xerox Network Systems extended characters, such as the bullet or dagger; see CharCodeTables, VirtualKeyboards in this manual) are ignored.
>
> Only one font is supported (though roman, italic, and bold typefaces do work).
>
> Graphics (lines, underlines, bitmaps) are ignored.
>
> Multiple column output does work.

### Set FX-80 Fast Mode

To set your default printer to be a fast mode FX-80, make the list (FASTFX80 FASTFX80) the CAR of the list DEFAULTPRINTINGHOST.

### Set FX-80 Destination

To set the default destination of all output to {LPT}.fastfx80, set the variable FASTFX80-DEFAULT-DESTINATION to an appropriate file name string. See the table above; the file name could be that of a regular file like {DSK}SPOOLED-FAST-OUTPUT.

### Set FX-80 Page Size

To set the driver's page size to match the paper in the printer, set the two variables \FASTFX80.INCHES-PER-PAGE (page height in inches) and \FASTFX80.INCHES-PER-LINE (page width in inches) to appropriate values. The defaults are 11 and 8.5, respectively. These can be set in your Lisp INIT file.

### Print a File

Select the HARDCOPY command from the background (right-button) menu. The system first formats the file for printing. Then, when the FX-80Driver actually starts transmitting to the printer, a small abort window, bearing the name of the document and the name of the printer, will appear near the top of your screen.

### Abort a Print Job

Clicking on the item marked ABORT in the print window will cleanly terminate the printing of the document.

Note: After aborting a print job, you may need to turn the printer off and back on to make sure that other files will print successfully.

## Printing in High-Quality Mode

Print in high-quality mode by sending output to the printer HQFX80, or by opening an image stream on a file with type HQFX80. High-quality mode printing supports all of Xerox Lisp's device-independent graphics operations, as well as multi-font printing and the XNS extended character set. It prints at 72 dot-per-inch resolution. Fidelity to the original document is better than in fast mode, though printing speed is slower.

### Set HQ Mode

To set your default printer to be a high-quality FX-80, make the list (HQFX80 HQFX80) the CAR of the list

DEFAULTPRINTINGHOST. You can use the PRINTERMENU module or your favorite structure editor to do this.

### Set Destination

To set the default destination of all output to {LPT}.hqfx80, set the variable HQFX80-DEFAULT-DESTINATION to an appropriate file namestring. This could be "{TTY}", "{RS232}", or even the name of a regular file like "{DSK}spooled-hq-output".

### Set Page Size

To set the driver's page size to match the paper in the printer, set the two variables \HQFX80.INCHES-PER-PAGE (page height in inches) and \HQFX80.INCHES-PER-LINE (page width in inches) to appropriate values. The defaults are 11 and 8.5, respectively. These can be set in your Lisp INIT file.

### Print a File

Select the HARDCOPY command. The system first formats the file for printing. Then, when the FX-80Driver actually starts transmitting to the printer, a small abort window, bearing the name of the document and the name of the printer, will appear near the top of your screen.

Note: After printing a document on HQFX80, you may need to turn the printer off and back on before you can print with FASTFX80 on that printer.

### Abort a Print Job

See above.

## FX Printer Compatibility

(FX80-PRINT *&KEY THING-TO-PRINT LANDSCAPE? COMPRESS? HIGH-QUALITY?*) [Function]

*THING-TO-PRINT* may be one of a window, bitmap, or file path name. If *THING-TO-PRINT* is a path name, the file will be treated as either a TEdit or Lisp source file, and printed in the appropriate style.

In the window or bitmap cases, *LANDSCAPE?* specifies landscape printing (X-coordinates run down the left margin) when non-NIL;

*COMPRESS?* specifies FX-80 compressed printing mode.

If *HIGH-QUALITY?* is non-NIL and *THING-TO-PRINT* is a path name, output will be sent to the default high-quality FX-80 printer, otherwise to the default fast FX-80 printer.

The *LANDSCAPE?*, *COMPRESS?*, and *HIGH-QUALITY?* arguments all default to NIL.

## Limitations

Landscape printing has not been implemented.

## Examples

Send text output to fast FX-80:

```
(SETQ FX-80 (OPENIMAGESTREAM "{LPT}.FASTFX80"))
(CL:FORMAT FX-80 "HELLO, WORLD~%%")
(CL:CLOSE FX-80)
```

Print source code on fast FX-80 (assuming the FastFX80 is not your default printer, but is on the list DEFAULTPRINTINGHOST):

```
(LISTFILES (HOST FASTFX80) "{DSK}MYPROGRAM")
```

Note: Source code is stored in pre-pretty-printed form on the file. The pretty-printer's default linelength (width of a line in characters) is normally 102, which is too wide for the FastFX-80s 8.5-inch wide page. To create source files which print nicely on the fast FX-80, you should set the variable FILELINELENGTH to a more appropriate value before you MAKEFILE. 70 works nicely on 8.5-inch paper with a standard font profile, though your mileage may vary.

Print source code in the the fast FX-80 mode, assuming the FastFX80 is your default printer:

```
(LISTFILES "{DSK}MYPROGRAM")
```

Print TEdit file in fast FX-80 mode, assuming the FastFX80 is your default printer:

```
(LISTFILES "{WAYCOOL:}<PUBLIC>GENSYM.TEDIT")
```

Print text and graphics in high-quality mode:

```
(SETQ FX-80 (OPENIMAGESTREAM "{LPT}" 'HQFX80))
(MOVETO 300 300 FX-80)
(CL:FORMAT FX-80 "HELLO, WORLD~%%")
(DRAWCIRCLE 300 300 230 '(ROUND 8) NIL FX-80)
(CL:CLOSE FX-80)
```

Print source code in high-quality mode, assuming the high-quality FX-80 is not your default printer, but is on the list DEFAULTPRINTINGHOST:

```
(LISTFILES (HOST HQFX80) "{DSK}MYPROGRAM")
```

Note: See the previous note regarding FILELINELENGTH and the fast FX-80. The same holds for high-quality FX-80 printing, and we recommend 70 as the value for FILELINELENGTH.

Print source code in high-quality mode, assuming the high-quality FX-80 is your default printer:

```
(LISTFILES "{DSK}MYPROGRAM")
```

Print TEdit file in high-quality mode, assuming the high-quality FX-80 is your default printer:

```
(LISTFILES "{WAYGNARLY:}<PUBLIC>MAGNUMOPUS.TEDIT")
```

[This page intentionally left blank]

# GCHAX

GCHax contains functions that are useful for tracking down storage leaks, i.e., objects that should be garbage but do not get garbage collected. There are functions for examining reference counts, locating pointers to objects, and finding circularities (which are among the chief culprits in storage leaks).

Typically, you might turn to GCHax when you notice that STORAGE claims there are more instances of a data type in use than you believe there should be.

## Installation

Load GCHAX.LCOM from the library.

## Functions

### Storage

The function STORAGE displays statistics on the amounts and distribution of the virtual memory space that has been allocated. If you suspect your program may have storage leaks (e.g., because (VMEMSIZE) keeps growing without obvious reason), this function is the place to start to get an indication of which kinds of objects are not being reclaimed. STORAGE is part of the standard Lisp sysout; you need not have loaded GCHAX to use it.

(STORAGE *TYPES PAGE-THRESHOLD IN-USE-THRESHOLD*) [Function]

With no arguments, STORAGE displays statistics for all data types, along with some summary information about the space remaining. The optional arguments let you refine the display.

If *TYPES* is given, STORAGE only lists statistics for those types. *TYPES* should be a type name or list of type names.

If *PAGE-THRESHOLD* is given, then STORAGE omits types that have fewer than *PAGE-THRESHOLD* pages allocated to them. The default *PAGE-THRESHOLD* is 2, so types that are not currently in use (consume no storage) do not appear unless you specify a *PAGE-THRESHOLD* of zero.

If *IN-USE-THRESHOLD* is given, then STORAGE omits types that have fewer than *IN-USE-THRESHOLD* instances in use (allocated and not yet freed).

For example, (STORAGE '(ARRAYP BITMAP)) lists only statistics for the types ARRAYP and BITMAP; (STORAGE NIL 6) lists only statistics for data types that have at least six pages allocated. (STORAGE NIL NIL 100) lists only statistics for data types that have at least 100 instances still in use.

The STORAGE function displays, for each Lisp data type, the amount of space allocated to the data type, and how much is currently in use. The display looks something like this:

```
Type          Assigned        Free items    In use     Total alloc
              pages [items]
FIXP            66    8448       7115        1333         447038
FLOATP          24    3072       2412         660         734877
LISTP         2574 ~298584       5294     ~293290        3545071
ARRAYP           8     512        245         267          48199
    . . .
```

Type — Is the name of the data type, as given to DATATYPE or the Common Lisp DEFSTRUCT.

Assigned — Is how much of your virtual memory is set aside for items of this type. Memory is allocated in quanta of two pages (1024 bytes). The numbers under Assigned show the number of pages and the total number of items that fit on those pages. The tilde (˜) on the LISTP line indicates that the number is approximate, since cdr-coding makes the precise counting of lists impossible—the amount of memory consumed by any particular list cell varies depending on its contents and how it was allocated.

Free items — Shows how many of the assigned items are available to be allocated (by the Interlisp *create* or the Common Lisp *make*-constructs); these constitute the free list for that data type.

In Use — Shows how many items of this type are currently in use, i.e., have pointers to them and hence have not been garbage collected. If this number is higher than your program seems to warrant, you may want to look for storage leaks. The sum of Free items and In Use is always the same as the total Assigned items.

Total Alloc — Is the total number of items of this type that you have ever allocated (created), or at least since the last call to BOXCOUNT that reset the counter.

STORAGE also prints some summary information about how much space is allocated and available collectively for fixed-length items (mainly data types, both user and built-in), variable-length items (arrays, bit maps, strings), and symbols. The variable-length items have fixed-length headers, which is why they also appear in the printout of fixed-length items. For example, the line printed for the data type BITMAP says how many bit maps have been allocated, but the figure displayed as "assigned pages" counts only the headers, not the space used by the variable-length part of the bitmap. The variable length portion is accounted in the summary statistics for "ArrayBlocks", where it is lumped with all other users of variable-length space, as it is not possible for the system to more finely discriminate the users of the space.

```
Data Spaces Summary
                                  Allocated          Remaining
                                    Pages              Pages
Datatypes (incl. LISTP etc.)        4370             \
ArrayBlocks (variable)              5770             -- 47758
ArrayBlocks (chunked)               2626             /
Symbols                             1000                 1048
```

```
variable-datum free list:
le 4           18 items;       72 cells.
le 16          84 items;      865 cells.
le 64          38 items;     1019 cells.
le 256         76 items;     7580 cells.
le 1024         2 items;     1548 cells.
le 4096        11 items;    18568 cells.
le 16384        1 items;     4864 cells.
others          2 items;    59565 cells.

Total cells free:    94081  total pages:  736
```

In the summary, Remaining Pages indicates how many more pages are available to be allocated to each type of datum. There is a single figure for both fixed- and variable-length objects, because they are allocated out of the same pool of storage.

Variable-length objects are allocated in two different ways, reflected in the items "variable" and "chunked." The distribution of the former among several different sized free lists is shown next.

## Storage Leak Tracking Functions

The functions in GCHax are oriented toward finding leaks that involve items of some data type not getting garbage collected.

There are two main kinds of leaks:

Items that are unintentionally being held onto.

Items that no user structure is pointing to but are not collected because of the nature of the garbage collector.

Examples of the former are structures assigned to global variables and left there after the program finishes.

Examples of the latter are principally circular structures — structures where you can follow a chain of pointers from an object that eventually returns to the same object. Circular lists, such as you get from (NCONC A A), are a special case of circular structures. See comments in "Limitations" below.

Note: All functions listed below have names beginning with \ to remind you that you are dealing with system internals, and to proceed with at least a little caution. Although these functions are generally safe, in that their casual use will not cause arbitrary damage, you certainly can produce unintended side effects.

In particular, the functions \SHOWGC and \COLLECTINUSE have modes in which they return a list of some kind of pointer; beware of unintentionally holding on to such a list (e.g., by having it get onto the history list), thereby preventing the eventual garbage collection of any of those pointers.

Useful for keeping values off the history list are the Executive command SHH for completely inhibiting history list entry, and the idiom (PROG1 NIL operation), e.g., (PROG1 NIL (INSPECT value)) to inspect a structure without holding on to a pointer to the inspect window. You may find it convenient to define your own Exec command to do inspection, e.g.,

```
(DEFCOMMAND IN (OBJ TYPE)
    (PROG1 NIL (INSPECT OBJ TYPE)))
```

The reference counts of all objects in the system are maintained in a global hash table, called the GC reference count table. Some or all of its contents can be viewed with the following function:

(\SHOWGC *ONLYTYPES COLLECT FILE CARLVL CDRLVL MINCNT*) [Function]

Displays on *FILE* (default T) all objects in the GC reference count table whose reference count is at least *MINCNT*, whose default value is 2.

If *ONLYTYPES* is given, it is a list of data type names to which \SHOWGC confines itself.

If *COLLECT* is T, \SHOWGC returns a list of all the objects it displays.

*CARLVL* and *CDRLVL* are print levels affecting the displaying of lists; they default to two and six, respectively. In the listing, collision entries in the reference count table are tagged with a *. Reference count operations on pointers in collision entries are much slower than on noncollision entries.

Objects with reference count of one (1) do not appear explicitly in the reference count table, so cannot be viewed with \SHOWGC, even if you set *MINCNT* as low as 1.

Note that if *COLLECT* is T, then the reference count of all the collected items is now one greater, due to the pointer to each from the returned list.

(\REFCNT *PTR*) [Function]

Returns the current reference count of *PTR*. Pointers that are not reference counted (e.g., symbols and small integers) are considered to have reference count 1. Since pointers from the stack (e.g., PROG variables) do not affect reference counts, it is possible for the reference count of an object to be zero without the object being garbage collected.

Note: If you call \REFCNT from the Common Lisp interpreter, e.g., by typing it at top-level, the answer is almost always too large by 1, as the interpreter itself holds reference-counted pointers to the arguments to the function it is calling. The same problem besets \FINDPOINTER (below). The problem does not exist from the Old Interlisp Exec, which uses the Interlisp interpreter. You can also avoid the problem by explicitly invoking the Interlisp interpreter; e.g.,

(EVAL '(\REFCNT *expression*)).

(\#COLLISIONS) [Function]

Returns a list of four elements:

Number of entries in the reference-count table, i.e., the number of objects in memory whose reference count is not 1;

Number of entries that are in collision chains;

Ratio of these numbers, i.e., the fraction of all entries that are in collision chains;

Ratio of the number of entries to the size of the hash table.

(\#OVERFLOWS) [Function]

Returns a list of four elements like \#COLLISIONS, but instead counts only objects whose reference count has overflowed (is greater than 62). Reference count operations on such objects are significantly slower than on other objects.

(\COLLECTINUSE *TYPE PRED)* [Function]

Is useful when (STORAGE *TYPE*) shows more objects in use than you think is right, but you can't find any such pointers yourself.

*TYPE* is a data type name or number other than LISTP. \COLLECTINUSE returns a list of all objects of that type that are thought to be in use, i.e., not free.

If *PRED* is supplied, it is a function of one argument. \COLLECTINUSE returns only objects for which *PRED* returns true. *PRED* must not allocate storage; you probably want it to be a compiled function.

Note: \COLLECTINUSE should be used with care. In a correctly functioning system, \COLLECTINUSE is generally safe. However, if the free list of *TYPE* has been smashed so that some free objects are not on it, this function can make matters much more confused, especially if the first 32-bit field of the data type in question contains a pointer field.

(\FINDPOINTER *PTR COLLECT/INSPECT? ALLFLG MARGIN ALLBACKFLG*) [Function]

Provides a brute-force approach to answering the question, "Who has a pointer to *x*?" \FINDPOINTER searches virtual memory, looking for places where *PTR* is stored. The search is not completely blind: unless *ALLFLG* is true, it does not look in places that cannot have reference-counted pointers, such as pname space or the stack. However, if the reference count of the object is zero, \FINDPOINTER searches the stack (and only the stack, if *ALLFLG* is NIL), since in this case there is no hope of finding pointers in the usual reference-counted spaces. If *ALLFLG* = :STACK, then \FINDPOINTER searches the stack in addition to places that contain reference-counted pointers, but not other unlikely places.

\FINDPOINTER prints out a description of each place *PTR* is found. If it is found in a list, it asks whether to recursively search for pointers to the list, so you can track lists back to a more identifying place, such as a symbol value cell or some data type. It recurses without asking if *ALLBACKFLG* is true. If *PTR* is found in a typed object, \FINDPOINTER names the field, if the data type declaration is available, and asks if you want to recursively search for pointers to this object. In either case, the search stops once enough places have been found to account for *PTR*'s reference count (unless *ALLFLG* is T).

If *COLLECT/INSPECT?* is true, \FINDPOINTER saves the identifiable pointers in a list. If *COLLECT/INSPECT?* = COLLECT, the list of pointers is returned as value; otherwise, it is offered for inspection.

*MARGIN* is the left margin (in units of characters) by which the reports of locations are initially indented. The default is zero. Recursive searches for pointers are indented relative to this position.

The current version does not know how to parse array space, so if *PTR* is found in an array, the best it can do is print the memory address where it found it, usually something of the form {}#nn,nnnnn. In addition, \FINDPOINTER doesn't even try to find *PTR* as a literal inside a compiled code object, since such references are not cell-aligned. Thus, \FINDPOINTER is really most helpful if the pointer is stored in fixed-length data space (e.g., in a field of a data type, or as the top-level value of a symbol); fortunately, this handles most of the interesting cases in practice.

Note: Of course, since it touches (potentially) a huge percentage of your virtual memory, \FINDPOINTER is completely disruptive of your working set.

(\FINDPOINTERS.OF.TYPE *TYPE FILTER*) [Function]

Calls \FINDPOINTER on each pointer in use of type *TYPE* that satisfies *FILTER*, a function of one argument, the pointer. A *FILTER* of NIL is considered the true predicate. *FILTER* can also be a list form to evaluate in which the variable PTR is used to refer to the pointer in question.

\FINDPOINTERS.OF.TYPE is essentially the same as

```
(for PTR in (\COLLECTINUSE TYPE)
     when <FILTER is satisfied>
     do (\FINDPOINTER PTR))
```

except that it takes care to discard the cells of the list returned from \COLLECTINUSE before calling \FINDPOINTER, to avoid seeing one extra reference per object.

For example,

```
(\FINDPOINTERS.OF.TYPE 'STREAM '(NOT (OPENP PTR)))
```

searches for pointers to all streams that are not currently open.

(\SHOW.CLOSED.WINDOWS) [Function]

Collects all windows that are not currently open or icons of open windows, then opens each window one by one.

For each window, you are prompted to press the left mouse button to close the window and go on to the next, or press right to do something different. In the latter case, you are prompted again to press the left button if you would like to search for pointers to the window, using \FINDPOINTER, or press the right button to just leave the window open on the screen and proceed.

Returns the total number of windows examined.

(\SHOWCIRCULARITY *OBJECT MAXLEVEL*) [Function]

Follows pointers from *OBJECT*. If it finds a path back to itself, it prints that path. This function is not exceptionally fast, and deliberately (for performance reasons) does not detect circularities in lists; it simply bottoms out on lists at *MAXLEVEL*, which defaults to 1,000. Circular lists are usually obvious enough anyway.

(\MAPGC *MAPFN INCLUDEZEROCNT*) [Function]

Maps over all entries in the GC reference count table, applying *MAPFN* to three arguments: the pointer, its reference count (an integer), and *COLLISIONP*, a flag that is T if the entry is a collision entry. Entries with reference count zero are not included unless *INCLUDEZEROCNT* is T. This function underlies \SHOWGC. Some care is required in the writing of *MAPFN*; it should try to minimize any reference-counting activity of its own, and in particular avoid anything that would decrement the reference count of the pointer passed to it.

## Limitations

GCHax is not very useful for finding ordinary circular lists, as the typical system has vast amounts of list structure, with nothing to distinguish the interesting ones.

However, if the circular list also contains instances of user data types, then those data types will tend to show up as overallocated, and hence amenable to the search functions in this module.

\FINDPOINTER does not know how to locate pointer arrays of more than 64 elements, so it is not helpful if a pointer you seek is located only in such an array.

[This page intentionally left blank]

# GRAPHER

Grapher contains a collection of functions and an interface for laying out, displaying, and editing graphs, that is, networks of nodes connected by links. Graphs have node labels but not link labels. Links are drawn by default as straight lines without arrowheads, but you can control the appearance of individual links. Node labels can be single lines of text, bitmaps of arbitrary size, or image objects. Facilities exist for calling functions at the nodes in a graph, and image objects containing graphs can be constructed so you can include graphs in documents and other image structures.

For instance, the Browser module uses graphs to represent function-calling structures (from MasterScope). Such a partially specified node list need have only the graph labels and the links specified. It is given to the LAYOUTGRAPH function along with some formatting information. LAYOUTGRAPH is a Grapher function which assigns a position to each node. There are formats for laying out trees, lattices, and cyclic graphs. LAYOUTGRAPH returns an instance of the GRAPH record, which is usually given to the function SHOWGRAPH. SHOWGRAPH displays a graph in a window.

## Installation

Load GRAPHER.LCOM from the library.

## User Interface

A typical way to use Grapher is to implement a function that creates a partially specified list of graph nodes representing some user data (or control) structure. Then you can use Grapher to display and manipulate or explore that structure.

Displayed graphs can be edited using the right button on the mouse. Nodes can be added, deleted, moved, enlarged, or shrunk. Links can be added or deleted.

Displayed graphs are often used as menus: selecting a node with the left or middle button can cause user-provided functions to be called on that node.

## Functions

Grapher functions perform the following tasks:

- Creating a graph.
- Laying out a graph for display.

- Displaying a graph.
- Editing a graph.
- Inserting a graph into a document.

These tasks are described in the following subsections. An additional subsection describes Grapher functions that perform other tasks.

## Creating a Graph

Start by creating a list of nodes for the graph. You can create them directly (see the section "GRAPHNODE Record"), or you can use the NODECREATE function.

(NODECREATE *ID LABEL POSITION TONODEIDS FROMNODEIDS FONT BORDER LABELSHADE*) [Function]

This function returns a GRAPHNODE record. The arguments of this function are the same as the corresponding fields of the GRAPHNODE record, as follows:

| Argument | GRAPHNODE Field |
|---|---|
| ID | NODEID |
| LABEL | NODELABEL |
| POSITION | NODEPOSITION |
| TONODEIDS | TONODE |
| FROMNODEIDS | FROMNODE |
| FONT | NODEFONT |
| BORDER | NODEBORDER |
| LABELSHADE | NODELABELSHADE |

*ID* and *LABEL* are required; *BORDER* defaults to 0, *LABELSHADE* defaults to WHITESHADE, and *POSITION* defaults to wherever it seems convenient.

You need to specify how the nodes are connected by providing a list of *TONODEID*S and *FROMNODEID*S for each node.

## Laying Out a Graph for Display

(LAYOUTGRAPH *NODELST ROOTIDS FORMAT FONT MOTHERD PERSONALD FAMILYD*) [Function]

Lays out a partially specified graph by assigning positions to its graph nodes. It returns a GRAPH record suitable for displaying with SHOWGRAPH. An example appears after the description of the arguments.

*NODELST* Is a list of partially specified GRAPHNODEs: only their NODELABEL, NODEID, and TONODE fields need to be filled in. NODEFONT fields may also contain font specifications to be used instead of the default supplied by the *FONT* argument. These optional fields are filled in appropriately if they are NIL. All other fields are ignored and/or overwritten.

*ROOTIDS* Is a list of the node identifiers of the nodes that become the roots.

The rest of the arguments are optional and control the format of the layout.

*FORMAT* Controls the layout of the graph. It is an unordered list of atoms or lists. The following options control the structure of the graph:

- COMPACT, the default, which lays out the graph as a forest (that is, a set of disjoint trees) using the minimal amount of screen space.
- FAST, which lays out the graph as a forest, sacrificing screen space for speed.
- LATTICE, which lays out the graph as a directed acyclic graph, that is, a lattice.

In addition, the following options control the direction of the graph:

- HORIZONTAL, the default, has roots at the left and links that run left-to-right.
- VERTICAL has roots at the top and links that run top-to-bottom.

The directions can be reversed by including the atom REVERSE in *FORMAT*.

Thus, for example,

- *FORMAT* = (LATTICE HORIZONTAL REVERSE) lays out horizontal lattices that have the roots on the right, with the links running right-to-left.
- *FORMAT* = (VERTICAL REVERSE) lays out vertical trees that have the roots at the bottom, with links running bottom-to-top.
- *FORMAT* = NIL lays out horizontal trees that have the roots on the left.

LAYOUTGRAPH creates virtual graph nodes to avoid drawing a tangle of messy lines in cases where the graph is not a forest or a lattice to begin with. It modifies the nodes of NODELST, which may involve changing some of the TONODEs fields to point to new nodes. The modified NODELST is set into the GRAPHNODEs field of a newly created GRAPH record, which is returned as the value of LAYOUTGRAPH. The creation of virtual nodes depends on whether LATTICE is a member of *FORMAT*.

In a forest, nodes are laid out by traversing the forest top-down, depth-first. If a node already has been laid out, LAYOUTGRAPH creates a copy of the node (the same NODELABEL, different NODEID, and no TONODEs), lays it down, and marks both it and the original node by setting their NODEBORDER fields and NODELABELSHADE fields. This occurs instead of drawing a link that might cut across arbitrary parts of the graph. Hence, a marked node occurs at least twice in the forest.

The default for marked nodes is to leave the shade alone and set the border to 1. To alter this appearance, add the (MARK . PROPS) to the FORMAT argument. PROPS is a property list. If it is NIL, marking is suppressed altogether. If it contains BORDER or LABELSHADE properties, those values are used in the corresponding fields of marked nodes. For example, a format of (MARK BORDER 5) would cause duplicated nodes to be boxed with borders 5 points wide.

*FORMAT* adds a few enhancements to this basic marking strategy, and can include one or both of these atoms:

- COPIES/ONLY—Only the new virtual nodes are marked. The original is left unmarked.
- NOT/LEAVES—Marking is suppressed when the node has no daughters.

For example,

- *FORMAT* = (COPIES/ONLY NOT/LEAVES) marks nodes that are copies of nodes that have daughters (for example, if you see a mark, the node has daughters that are not drawn).
- FORMAT = (NOT/LEAVES) marks both copies and originals, but only when they have daughters.
- *FORMAT* = NIL marks originals and copies regardless of progeny.

If *FORMAT* includes LATTICE, then a node that is the daughter of more than one node is not marked. Instead, links from all its parents are drawn to it. No attempt is made to avoid drawing lines through nodes or to minimize line crossings. However, in HORIZONTAL format, nodes are positioned so that From is always left of To.

Similar conventions hold for the other formats. In VERTICAL format, for instance, the TONODEs of a node are positioned beneath it, and the FROMNODEs are positioned above it.

Cyclic graphs cannot be drawn using this convention, since a node cannot be left of itself. When LAYOUTGRAPH detects a node that points to itself, directly or indirectly, it creates a virtual node, as described above, and marks both the original and the copy. If *FORMAT* includes COPIES/ONLY, then only the newly created node is marked.

*FONT* Is a font specification for use as the default NODEFONT.

The remaining three arguments control the distances between nodes. NILs cause "pretty" defaults based on the size of *FONT*.

*PERSONALD* Is specified in points; it controls the minimum distance between any two nodes.

*MOTHERD* Is the minimum distance between a mother and her daughters.

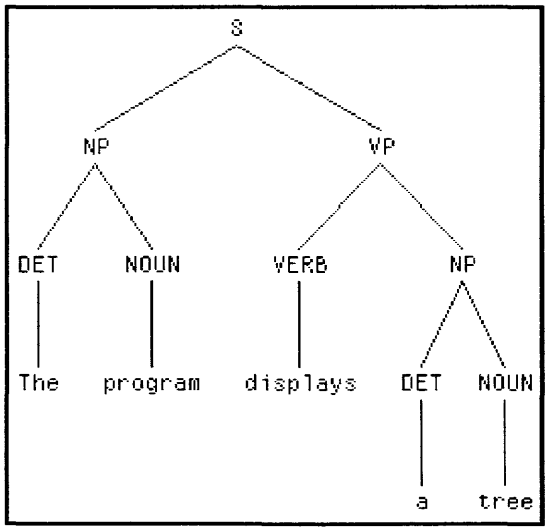
*FAMILYD* Controls the minimum distance between nodes from different nuclear families. The closest two sister nodes can be is *PERSONALD*. The closest that two nodes that are not sisters can be is *PERSONALD* + *FAMILYD*.

LAYOUTGRAPH reads but does not change the fields NODEBORDER and NODELABELSHADE of the nodes given it. The marked nodes are an exception. Thus, if you plan to install black borders around the nodes after the nodes have been laid out (for example, by RESET/NODE/BORDER, described in the section "Performing Other Tasks"), it is a good idea to give LAYOUTGRAPH nodes that have white borders. This causes the nodes to be laid out far enough apart that when you blacken the borders later, the labels of adjacent nodes are not overwritten.

As an example, to create and display the following parse tree for the sentence "The program displays a tree.", enter

```
(SETQ Snode (NODECREATE 'S 'S NIL '(NP1 VP)))
(SETQ NP1node (NODECREATE 'NP1 'NP NIL '(DET1 NOUN1) '(S) ))
(SETQ DET1node (NODECREATE 'DET1 'DET NIL '(THE) '(NP1) ))
(SETQ THEnode (NODECREATE 'THE 'The NIL NIL '(DET1) ))
(SETQ NOUN1node (NODECREATE 'NOUN1 'NOUN NIL '(PROGRAM) '(NP1) ))
(SETQ PROGRAMnode (NODECREATE 'PROGRAM 'program NIL NIL '(NOUN1) ))
(SETQ VPnode (NODECREATE 'VP 'VP NIL '(VERB NP2) '(S) ))
(SETQ VERBnode (NODECREATE 'VERB 'VERB NIL '(DISPLAYS) '(VP) ))
(SETQ DISPLAYSnode (NODECREATE 'DISPLAYS 'displays NIL NIL '(VERB) ))
(SETQ NP2node (NODECREATE 'NP2 'NP NIL '(DET2 NOUN2) '(VP) ))
(SETQ DET2node (NODECREATE 'DET2 'DET NIL '(A) '(NP2) ))
(SETQ Anode (NODECREATE 'A 'a NIL NIL '(DET2) ))
(SETQ NOUN2node (NODECREATE 'NOUN2 'NOUN NIL '(TREE) '(NP2) ))
(SETQ TREEnode (NODECREATE 'TREE 'tree NIL NIL '(NOUN2) ))

(SHOWGRAPH (LAYOUTGRAPH (LIST Snode NP1node DET1node THEnode NOUN1node
PROGRAMnode VPnode VERBnode DISPLAYSnode NP2node DET2node Anode
NOUN2node TREEnode ) '(S) '(VERTICAL)))
```



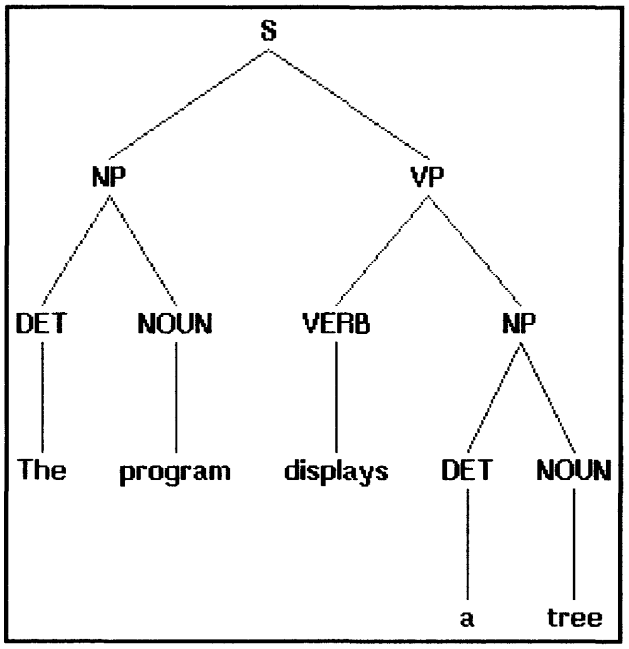
(LAYOUTSEXPR *SEXPR FORMAT BOXING FONT MOTHERD PERSONALD FAMILYD*) [Function]

Is just like LAYOUTGRAPH, except it gets its graph as an s-expression rather than a list of GRAPHNODEs. Its first argument is recursively interpreted as follows: If the s-expression is a non-list, its NODELABEL is itself and it has no TONODEs; else its CAR is taken as its NODELABEL and its CDR, which must be a list of s-expressions, is taken as its TONODEs.

Note: Circular s-expressions are allowed.

For example, to display the following parse tree for the sentence "The program displays a tree.", enter:

```
[SHOWGRAPH (LAYOUTSEXPR '(S (NP (DET The)(NOUN program))
(VP (VERB displays) (NP (DET a)(NOUN tree)))) '(VERTICAL)
NIL '(HELVETICA 12 BRR]
```



## Displaying a Graph

(SHOWGRAPH *GRAPH W LEFTBUTTONFN MIDDLEBUTTONFN TOPJUSTIFYFLG ALLOWEDITFLG COPYBUTTONEVENTFN*) [Function]

Displays the nodes in *GRAPH*.

If *W* is a window, the graph is displayed in it. If the graph is larger than the window, the window is made a scrolling window. If *W* is NIL, the graph is displayed in a window large enough to hold it. If *W* is a string, the graph is displayed in a window large enough to hold it, and the window uses the string for the window title. The graph is stored on the GRAPH property of the window. SHOWGRAPH returns the window.

If either *LEFTBUTTONFN* or *MIDDLEBUTTONFN* is non-NIL, the window is given a *BUTTONEVENTFN* that, in effect, turns the graph into a menu. Whenever you press left or middle mouse button and the cursor is over a node, that node is displayed inverted, indicating that it is selected. Releasing the mouse button calls either the *LEFTBUTTONFN* or the *MIDDLEBUTTONFN* with two arguments: the selected node and the window. The node is a GRAPHNODE, or NIL if the cursor was not over a node when the button was released. The function can access the graph via the window's GRAPH property.

The graph's initial position in the window is determined by *TOPJUSTIFYFL*. If T, the graph's top edge is positioned at the top edge of the window; if NIL, the graph's bottom edge is positioned at the bottom edge of the window.

*ALLOWEDITFLG* and *COPYBOTTONFLG* are described under "Editing a Graph," below.

Note: The node labels are reprinted whenever the graph is redisplayed. If this makes scrolling of a large graph unacceptably slow, some speedup may be achieved by instructing Grapher to cache bitmaps of the labels with the nodes so they can be rapidly BITBLTed to the screen (set the variable CACHE/NODE/LABEL/BITMAPS to T). The possible gain in time, however, may be offset by the increased storage required for the cached bitmaps.

(DISPLAYGRAPH *GRAPH STREAM CLIP/REG TRANS*) [Function]

Displays the specified graph on *STREAM*, which can be any image stream, with coordinates translated to *TRANS*. Some streams might also implement *CLIP/REG* as a clipping region. This is primarily to improve efficiency for the display.

(HARDCOPYGRAPH *GRAPH/WINDOW FILE IMAGETYPE TRANS*) [Function]

Produces a file containing an image of *GRAPH*, that is, like SHOWGRAPH, but for files. If *GRAPH/WINDOW* is a window, HARDCOPYGRAPH operates on its GRAPH window property. The *FILE* and *IMAGETYPE* argument are given to OPENIMAGESTREAM to obtain a stream that the graph will be displayed on. *TRANS* is the position in screen points (that is, it is scaled by the image stream's DSPSCALE) of the lower-left corner of the graph relative to the lower-left corner of the piece of paper.

GRAPH/HARDCOPY/FORMAT [Variable]

Is used to control the format of the graph when printing to paper. It is a property list that contains the following properties.

- MODE

  Determines the orientation of the hardcopy of the graph. The value can be LANDSCAPE, or PORTRAIT (the default). If LANDSCAPE, the graph is shown with the longer paper edge as the major axis. If PORTRAIT, graph is shown with the shorter paper edge as the major axis. If you use the window menu command to hardcopy, the graph is shown in PORTRAIT mode.

- PAGENUMBERS

  Determines whether to print the page number. The value can be T or NIL. If T, GRAPHER prints the page number in X-Y format on the upper right corner of each page. If NIL, no page number is printed.

- TRANS

  Determines where to position the graph on paper. The value can be NIL or a position. If NIL, each graph is positioned at the center of the paper. If a position, GRAPHER determines the location in screen points of the lower left corner of the graph relative to the lower left corner of the paper.

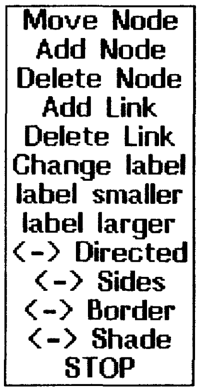
The initial value of GRAPH/HARDCOPY/FORMAT is set to (MODE PORTRAIT PAGENUMBERS T TRANS NIL)

DEFAULT.GRAPH.WINDOWSIZE [Variable]

Contains a list of two numbers in screen points. The first number indicates the window width. The second number indicates the window height. This variable is used to control the maximum size of a graph window when it first gets displayed.

## Editing a Graph

If *ALLOWEDITFLG* in the SHOWGRAPH function is non-NIL, you can position the cursor over a node and use the right mouse button to edit the graph. (The normal window commands can be accessed by right-clicking (pressing the right mouse button) in the border or title regions.) Holding down the control key and simultaneously pressing the right mouse button allows you to position nodes by tracking the cursor. Pressing the right mouse button without the control key pops up the following menu of edit operations.



The edit operations allow moving, adding, and deleting of nodes and links.

- Adding a node prompts for a NODELABEL, creates a new node with that label, adds it to the graph, and allows you to position it.
- Deleting a node removes it (using DREMOVE) from the graph after deleting all of the links to and from it.
- Selecting Directed or Sides allows you to control how links are drawn between nodes. See the section "Graph Record" for more information.

- Selecting Border allows you to invert the border around a node's label.
- Selecting Shades allows you to invert a node's label.

When you select the STOP menu command, the graph window is closed.

*COPYBUTTONEVENTFN* is a function to be run when you copy-select from the Grapher window. If this is not specified, the default simply COPYINSERTs a Grapher image object.

Certain fields of the GRAPH record contain functions that are called from the graph editor menu to perform actions on an element in the displayed graph. They allow the graph to serve as a simple edit interface to the structure being graphed.

The following fields of a graph contain editing functions and the arguments that are passed to those functions when they are called. In all cases, *GRAPH* is the graph being displayed, and *WINDOW* is the window in which it is displayed.

GRAPH.MOVENODEFN *NODE NEWPOS GRAPH WINDOW OLDPOS* [Record field]

Contains a function that is called with the arguments shown after you have stopped moving a node interactively; that is, it is not called as the node is being moved. *NEWPOS* is the new position of the node, *OLDPOS* its original position. The difference between them can be used, for example, to move other related nodes by the same distance.

GRAPH.ADDNODEFN *GRAPH WINDOW* [Record field]

Is called when you select ADD A NODE. Returns a node, or NIL if no new node is to be added. A node-moving operation is called on the new node after it is created to determine its position.

GRAPH.DELETENODEFN *NODE GRAPH WINDOW* [Record field]

Is called when a node is deleted. Before this function is called, all of the links to or from the node are deleted.

GRAPH.ADDLINKFN *FROM TO GRAPH WINDOW* [Record field]

Is called when a link is added.

GRAPH.DELETELINKFN *FROM TO GRAPH WINDOW* [Record field]

Is called when a link is deleted, which can be either directly or from deleting a node.

GRAPH.FONTCHANGEFN *HOW NODE GRAPH WINDOW* [Record field]

Is called for side effect only when you ask for the label on a node to be made larger or smaller. *HOW* is either LARGER or SMALLER.

GRAPH.CHANGELABELFN *GRAPH NODE* [Record field]

Is called for side effect only when you ask to change the label of a node, for example, using EDITGRAPH.

GRAPH.INVERTBORDERFN *NODE GRAPH* [Record field]

Is called for side effect only when you ask to invert the border of a node, for example, using EDITGRAPH.

GRAPH.INVERTLABELFN *NODE GRAPH* [Record field]

Is called for side effect only when you ask to invert the label of a node, for example, using EDITGRAPH.

### Editing Menu

The editing menu is controlled by two variables.

EDITGRAPHMENU [Variable]

Contains the editing menu, if it exists, or NIL. If you press the right button and it is NIL, a fresh menu will be created from EDITGRAPHMENUCOMMANDS.

EDITGRAPHMENUCOMMANDS [Variable]

A list of menu items used to create EDITGRAPHMENU. The contents of EDITGRAPHMENUCOMMANDS must be a list that can be used as the ITEMS field of a MENU; see MENU in the *Interlisp-D Reference Manual* for details.

## Inserting a Graph into a TEdit Document

A graph data structure can be encapsulated in a Grapher image object so that it can be inserted in a TEdit document or other image structure. Grapher image objects are constructed by the following function.

(GRAPHEROBJ *GRAPH HALIGN VALIGN*) [Function]

Returns a Grapher-type image object that displays *GRAPH*.

*HALIGN* and *VALIGN* specify how the graph is to be aligned with respect to the reference point in its host, for example, a TEdit file or image object window. They can be numbers between zero and one, specifying as a proportion of the width/height of the graph the point in the graph that overlays the reference point; zero means that the graph will sit completely above and to the left of the reference point, and one means it will sit completely below and to the right.

They can also be pairs of the form (*NODESPEC POS*), where

*NODESPEC* specifies a node that the graph is to be aligned by, and *POS* specifies where in the node the alignment point is. The *NODESPEC* can be either a NODEID or one of the atoms *TOP*, *BOTTOM*, *LEFT*, or *RIGHT*, indicating the topmost, bottommost, etc., node of the graph.

*POS* can be a number specifying proportional distances from the lower-left corner of the node, or the atom BASELINE, indicating the character baseline (for *VALIGN*, or simply zero for *HALIGN*).

For example, to align a linguistic tree so that the baseline of the root node is at the reference point, *VALIGN* is (*TOP* BASELINE).

The *BUTTONEVENTINFN* of the image object pops up a single-item menu, which, if selected, causes the graph editor to be run.

## Performing Other Tasks

Grapher functions also allow you to return the smallest region containing all nodes, invert a node region, reset fields in a node, print a graph to a stream, read a graph from a stream, and edit a graph.

(GRAPHREGION *GRAPH*) [Function]

Returns the smallest region containing all of the nodes in GRAPH.

(FLIPNODE *NODE DS*) [Function]

Inverts a region in the stream DS that is one pixel bigger all around than NODE's region. This makes it possible to see black borders after the node has been flipped.

(RESET/NODE/BORDER *NODE BORDER STREAM GRAPH*) [Function]

and

(RESET/NODE/LABELSHADE *NODE SHADE STREAM*) [Function]

Reset the appropriate fields in the node. If *STREAM* is a display stream or a window, the old node is erased and the new node is displayed. Changing the border may change the size of the node, in which case the lines to and from the node are redrawn. The entire graph must be available to RESET/NODE/BORDER for this purpose, either supplied as the *GRAPH* argument or obtained from the GRAPH property of *STREAM,* if it is a window. Both functions take the atom INVERT as a special value for *BORDER* and *SHADE*. They read the node's current border or shade, calculate what is needed to invert it, and do so.

(DUMPGRAPH *GRAPH STREAM*) [Function]

Prints *GRAPH* out on *STREAM* in a special, relatively compact encoding that can be interpreted by the function READGRAPH, below. Graphs cannot be saved on files simply by ordinary print functions such as PRIN2. This is because the Grapher functions use FASSOC (that is, EQ, not EQUAL) to fetch a graph node given its ID, so reading it back in gives the right result only if the IDs are atomic. HPRINT resolves this problem, but it tends to dump too much information: it dumps a complete description of the node font, for example, including the character bit maps. DUMPGRAPH and READGRAPH are used in the implementation of Grapher image objects.

(READGRAPH *STREAM*) [Function]

Reads information from *STREAM* starting at the current file pointer and returns a graph structure equivalent to the one that was given to DUMPGRAPH.

(EDITGRAPH *GRAPH WINDOW*) [Function]

Enables editing *GRAPH* in *WINDOW*. If *GRAPH* is NIL, an empty graph is created for editing. If *WINDOW* is NIL, a window of appropriate size is created.

(GRAPHERPROP *GRAPH PROP NEWVALUE*) [Function]

Accesses GRAPH.PROPS field of *GRAPH* record. The function returns the previous value of *GRAPH*'s *PROP* aspect. If *NEWVALUE* is given, it is stored as the new *PROP* aspect.

## Grapher Record Structure

Grapher has GRAPH records which represent graphs. Within these records are GRAPHNODE records which are lists of graph records.

### GRAPH Record

A graph is represented by a GRAPH record, which has the following fields:

GRAPHNODE
DIRECTEDFLG
SIDESFLG
GRAPH.MOVENODEFN
GRAPH.ADDNODEFN
GRAPH.DELETENODEFN
GRAPH.ADDLINKFN
GRAPH.DELETELINKFN
GRAPH.CHANGELABELFN
GRAPH.INVERTBORDERFN
GRAPH.INVERTLABELFN
GRAPH.FONTCHANGEFN
GRAPH.PROPS

GRAPHNODE is a list of graph nodes, and is described below.

DIRECTEDFLG and SIDESFLG are flags that control how links are drawn between the nodes. If DIRECTEDFLG is NIL, Grapher draws each link in such a way that it does not cross the node labels of the nodes it runs between. Often, this leaves some ambiguity, which is settled by SIDESFLG. If SIDESFLG is NIL, Grapher prefers to draw links that go between the top and bottom edges of nodes. If SIDESFLG is non-NIL, Grapher prefers to draw links between the sides of the nodes.

If DIRECTEDFLG is non-NIL, the edges are fixed, for example, always to the left edge of the To node. This can cause links to cross the labels of the nodes they run between. In this case, if SIDESFLG is NIL, the From end of the link is attached to the bottom edge of the From node; the To end of the link is attached to the top edge of the To node. If DIRECTEDFLG is non-NIL and SIDESFLG is non-NIL, the From end of the link is attached to the right edge of the From node; the To end of the link is attached to the left edge of the To node.

GRAPH.PROPS is a list in property-list format, and is accessed by the function GRAPHERPROP.

The remaining fields give you hooks into the graph editor, and are described in the section "Editing a Graph".

## GRAPHNODE Record

The GRAPHNODE record has the following fields of interest:

NODELABEL Is what gets displayed as the node. If this is a bit map, BITBLT is used; if it is an image object, its *IMAGEBOXFN* and *DISPLAYFN* are used. Anything else is printed with PRIN3; see the *Interlisp-D Reference Manual*. Image objects can be used to give a node a larger-than-normal margin around its text label.

NODEID Is a unique identifier. NODEIDs are used in the link fields instead of pointers to the nodes themselves, so that circular Lisp structures can be avoided. NODEIDs are often used as pointers to the structure represented by the graph.

TONODE Is a list of NODEIDs. A link runs from the currently selected node to each node in TONODEs. Entries in this field can be used to specify properties of the lines drawn between nodes.

If an item in the TONODEs of the current node N1 is not a NODEID but rather a list of the form:

(LINK% PARAMETERS *TONODEID . PARAMLIST)*

then *PARAMLIST* is interpreted as a property list specifying properties of the link drawn from N1 to *TONODEID*.

Properties of *PARAMLIST* currently noticed are LINEWIDTH, DASHING, COLOR, and DRAWLINKFN. The first three are passed directly to DRAWLINE.

For example, if the TONODEs for A is:

```
((LINK% PARAMETERS B LINEWIDTH 4 DASHING (3 3))
(LINK% PARAMETERS C DASHING (5 1) COLOR 12))
```

then two dashed lines will emanate from A, with the one to B having width 4 and dashing (3 3), and the one to C having the default width 1, dashing (5 1), and color (if implemented) 12.

If the property DRAWLINKFN is on the list, then its value must be a function to be called instead of DRAWLINE. It is passed all the arguments of DRAWLINE plus the *PARAMLIST* as a last argument.

For convenience, the variable LINKPARAMS is set to the constant value LINK% PARAMETERS. When DISPLAYGRAPH scales the graph to the units of a particular output stream, the properties whose names are found on SPECVAR *SCALABLELINKPARAMETERS* are also scaled.

| | |
|---|---|
| FROMNODE | Is a list of NODEIDs. A link runs to the currently selected node from each node in FROMNODEs. |
| NODEPOSITION | Is the location of the center of the node (a POSITION). |
| NODEFONT | Specifies the font in which this node's label is displayed. It can be any font specification acceptable to FONTCREATE, including a FONTDESCRIPTOR. NODEFONT is changed by the graph edit operations Larger and Smaller. When this happens, the font family may be changed as well as the size. Default is the value of DEFAULT.GRAPH.NODEFONT (initially NIL, which specifies the system DEFAULTFONT). |
| NODEBORDER | Specifies the shade and width of the border around a node via the following values: |

| | |
|---|---|
| NIL,0 | No border; equals border of width zero |
| T | Black border, one pixel wide |
| 1,2,3. . . | Black border of the given width |
| -1,-2. . . | White border of the given width |
| (W S) | Where W is an integer and S is a texture or a shade; yields a border W pixels wide filled with the given shade S; see the *Interlisp-D Referene Manual*. |

Default is the value of DEFAULT.GRAPH.NODEBORDER (initially NIL).

| | |
|---|---|
| NODELABELSHADE | Contains the background shade of the node. If this field is non-NIL, then when a node is displayed, the label area for the node is first painted as specified by this field, then the label is printed in INVERT mode. This does not apply to labels that are bit maps or image objects. The legal values for the field are: NIL (same as WHITESHADE), T (same as BLACKSHADE), a texture, or a bit map. Default is the value of DEFAULT.GRAPH.NODELABELSHADE, which is initially NIL. |
| NODEWIDTH and NODEHEIGHT | Are initially set by Grapher to be the width and height of the node's NODELABEL. |

## Limitations

Grapher does not work well with packages. Because the node labels are printed with PRIN3, Grapher does not visually distinguish nodes whose labels are symbols in different packages; you do not see that fact displayed in the graph.

# GRAPHZOOM

GraphZoom allows you to work with graphs (from Grapher) at different scales.

## Requirements

GRAPHER

## Installation

Load GRAPHER.LCOM and GRAPHZOOM .LCOM from the library.

## User Interface

The interface to GraphZoom is through the function SHOWZOOMGRAPH which is used instead of SHOWGRAPH. The resulting grapher window has the capability of being zoomed by using a menu that is attached to the top. To use, call SHOWZOOMGRAPH:

The window will have a menu above it that contains the items LARGER and SMALLER. Selecting in the menu with the left button will cause a small change in scale; selecting with the middle button will cause a larger change in scale.

The graph will zoom toward the center of the window.

## Function

(SHOWZOOMGRAPH *GRAPH WINDOW LEFTBUTTONFN MIDDLEBUTTONFN TOPJUSTIFYFLG ALLOWEDITFLG INITSCALE*)

[Function]

The arguments are the same as in the Grapher function SHOWGRAPH. *INITSCALE* is the initial scale at which the graph is shown. The default, 1.0, is the same as in SHOWGRAPH. Larger scales make the graph appear smaller.

## Limitations

The zooming of text is driven from a list of font descriptors stored as the value of the variable DECREASING.FONT.LIST. The values on this list are Times Roman 72, 36, 30, Helvetica 24, 18, 14, 12, 10, 8, 5, 4, and 3. Graphs that have fonts other than Helvetica will get printed in the closest Helvetica size. This is often smaller than the corresponding Gacha font.

# HASH

Note: This module is provided for backwards compatibility. New applications should use the HASH-FILE Library Module instead of this module.

Hash is a hash-coded dictionary facility, providing much the same functionality as hash arrays do, except that the data is stored in a file.

Hash permits information associated with string or atom keys to be stored on and retrieved from files. The information (or values) associated with the keys in a file may be numbers, strings, or arbitrary Lisp expressions. The associations are maintained by a hashing scheme that minimizes the number of file operations it takes to access a value from its key.

Information is saved in a hash file, which is analogous to a hash array. Actually, hash file can be either the file itself, or the handle on that file which is used by the Hash module. The latter, of data type HashFile, is the datum returned by CREATEHASHFILE or OPENHASHFILE, currently an array record containing the hash file name, the number of slots in the file, the used slots, and other details. All other functions with hash file arguments use this datum.

In older implementations (e.g., for Interlisp-10), hash files came in several varieties, according to the types of value stored in them. The EMYCIN system provided even more flexibility.

This system only supports the most general EXPR type of hash files and EMYCIN-style TEXT entries, in the same file. The VALUETYPE and ITEMLENGTH arguments are for the most part ignored. Two-key hashing is supported in this system but is discouraged as it is only used in EMYCIN, not in the Interlisp-10 system. The functions GETPAGE, DELPAGE, and GETPNAME, which manipulate secret pages, do not exist in this implementation. However, it is permissible to write data at the end of a hash file; that data will be ignored by the Hash module, and can be used to store additional data.

The Hash module views files as a sequence of bytes, randomly accessible. No notice is made of pages, and it is assumed that the host computer buffers I/O sufficiently.

Hash files consist of a short header section (8 bytes), a layer of pointers (4*HASHFILE:Size bytes) followed by ASCII data. Pointers are 3 bytes wide, preceded by a status byte. The pointers point to key PNAMES in the data section, where each key is followed by its value.

Deleted key pointers are reused but deleted data space is not, so rehashing is required if many items have been replaced.

The data section starts at 4*HASHFILE:Size + 9, and consists of alternating keys and values. As deleted data is not rewritten, not all data in the data section is valid.

When a key hashes into a used slot, a probe value is added to it to find the next slot to search. The probe value is a small prime derived from the original hash key.

## Requirements

Hash files must reside on a random-access device (not a TCP/IP file server).

## Installation

Load HASH.LCOM from the library.

## Functions

### Creating a Hash File

(CREATEHASHFILE *FILE VALUETYPE ITEMLENGTH #ENTRIES SMASH COPYFN*) [Function]

Creates a new hash file named *FILE*. All other arguments are optional.

*VALUETYPE* is ignored in this implementation; any hash file can accommodate both Lisp expressions and text.

*ITEMLENGTH* is not used by the system but is currently saved on the file (if less than 256) for future use.

*#ENTRIES* is an estimate of the number of entries the file will have. (This should be a realistic guess.)

*SMASH* is a hash file datum to reuse.

*COPYFN* is a function to be applied to entries when the file is rehashed (see the description of REHASHFILE, below).

### Opening and Closing Hash Files

Before you can use a hash file with this module, you have to open it using the function

(OPENHASHFILE *FILE ACCESS ITEMLENGTH #ENTRIES SMASH*) [Function]

Reopens the previously existing hash file *FILE*.

*Access* may be INPUT (or NIL), in which case *FILE* is opened for reading only, or BOTH, in which case *FILE* is open for both input and output. Causes an error "not a hashfile", if *FILE* is not recognized as a hash file.

*ITEMLENGTH* and *#ENTRIES* are for backward compatibility with EMYCIN where OPENHASHFILE also created new hash files; these arguments should be avoided.

*SMASH* is a hash file datum to reuse.

If *ACCESS* is BOTH and *FILE* is a hash file open for reading only, OPENHASHFILE attempts to close it and reopen it for writing. Otherwise, if *FILE* designates an already open hash file, OPENHASHFILE is a no-op.

OPENHASHFILE returns a hash file datum.

(CLOSEHASHFILE *HASHFILE REOPEN*) [Function]

Closes *HASHFILE* (when you are finished using a hash file, you should close it). If *REOPEN* is non-NIL it should be one of the accepted access types. In this case the file is closed and then immediately reopened with *ACCESS* = *REOPEN*. This is used to make sure the hash file is valid on the disk.

## Storing and Retrieving Data

(PUTHASHFILE *KEY VALUE HASHFILE KEY2*) [Function]

Puts *VALUE* under *KEY* in *HASHFILE*. If *VALUE* is NIL, any previous entry for *KEY* is deleted. *KEY2* is for EMYCIN two-key hashing: *KEY2* is internally appended to *KEY* and they are treated as a single key.

(GETHASHFILE *KEY HASHFILE KEY2*) [Function]

Gets the value stored under *KEY IN HASHFILE*. *KEY2* is necessary if it was supplied to PUTHASHFILE.

(LOOKUPHASHFILE *KEY VALUE HASHFILE CALLTYPE KEY2*) [Function]

A generalized entry for inserting and retrieving values; provides certain options not available with GETHASHFILE or PUTHASHFILE. LOOKUPHASHFILE looks up *KEY* in *HASHFILE*.

*CALLTYPE* is an atom or a list of atoms. The keywords are interpreted as follows:

| | |
|---|---|
| RETRIEVE | If *KEY* is found, then if *CALLTYPE* is or contains RETRIEVE, the old value is returned from LOOKUPHASHFILE; otherwise returns T. |
| DELETE | If *CALLTYPE* is or contains DELETE, the value associated with *KEY* is deleted from the file. |
| REPLACE | If *CALLTYPE* is or contains REPLACE, the old value is replaced with *VALUE*. |
| INSERT | If *CALLTYPE* is or contains INSERT, LOOKUPHASHFILE inserts *VALUE* as the value associated with *KEY*. |

Combinations are possible.

For example, (RETRIEVE DELETE) will delete a key and return the old value.

(PUTHASHTEXT *KEY SRCFIL HASHFILE START END*) [Function]

Puts text from stream *SRCFIL* onto *HASHFILE* under *KEY*. *START* and *END* are passed directly to COPYBYTES.

(GETHASHTEXT *KEY HASHFILE DSTFIL*) [Function]

Uses COPYBYTES to retrieve text stored under *KEY* on *HASHFILE*. The bytes are output to the stream *DSTFIL*.

## Functions for Manipulating Hash Files

(HASHFILEP *HASHFILE WRITE?*) [Function]

Returns *HASHFILE* if it is a valid, open hash file datum or returns the hash file datum associated with *HASHFILE* if it is the name of an open hash file. If *WRITE?* is non-NIL, *HASHFILE* must also be open for write access.

(HASHFILEPROP *HASHFILE PROPERTY*) [Function]

Returns the value of a *PROPERTY* of a *HASHFILE* datum. Currently accepted properties are NAME, ACCESS, VALUETYPE, ITEMLENGTH, SIZE, #ENTRIES, COPYFN and STREAM.

(HASHFILENAME *HASHFILE*) [Function]

Same as (HASHFILEPROP *HASHFILE* 'NAME).

(MAPHASHFILE *HASHFILE MAPFN DOUBLE*) [Function]

Maps over *HASHFILE* applying *MAPFN*. If *MAPFN* takes two arguments, it is applied to *KEY* and *VALUE*. If *MAPFN* only takes one argument, it is only applied to *KEY* and saves the cost of reading the value from the file. If *DOUBLE* is non-NIL, then *MAPFN* is applied to *(KEY1 KEY2 VALUE*) or (*KEY1 KEY2*) if the *MAPFN* only takes two arguments.

(REHASHFILE *HASHFILE NEWNAME*) [Function]

As keys are replaced, space in the data section of the file is not reused (though space in the key section is). Eventually the file may need rehashing to reclaim the wasted data space. REHASHFILE is really a special case of COPYHASHFILE, and creates a new file. If *NEWNAME* is non-NIL, it is taken as the name of the rehashed file.

The system automatically rehashes files when 7/8 of the key section is filled. The system will print a message when automatically rehashing a file if the global variable REHASHGAG is non-NIL.

Certain applications save data outside Hash's normal framework. Hash files for those applications will need a custom *COPYFN* (supplied in the call to CREATEHASHFILE), which is used to copy data during the rehashing process. The COPYFN is used as the *FN* argument to COPYHASHFILE during the rehashing.

(COPYHASHFILE *HASHFILE NEWNAME FN VALUETYPE LEAVEOPEN*) [Function]

Makes a copy of *HASHFILE* under *NEWNAME*.

Each key and value pair is moved individually and if *FN* is supplied, is applied to *(KEY VALUE HASHFILE NEWHASHFILE*).

What it returns is used as the value of the key in the new hash file. (This lets you intervene, perhaps to copy out-of-band data associated with *VALUE*.)

*VALUETYPE* is a no-op.

If *LEAVEOPEN* is non-NIL then the new hash file datum is returned open, otherwise the new Hash file is closed and the name is returned.

(HASHFILESPLST *HASHFILE XWORD*) [Function]

Returns a Lisp generator for the keys in *HASHFILE*, usable with the spelling corrector. If *XWORD* is supplied, only keys starting with the prefix in *XWORD* are generated.

## Global Variables of Hash

HASHFILEDEFAULTSIZE [Variable]

Size used when *#ENTRIES* is omitted or is too small. Default is 512.

HASHFILERDTBL [Variable]

The hash file read table. Default is ORIG.

HASHLOADFACTOR [Variable]

The ratio, used slots/total slots, at which the system rehashes the file. Default is 0.875.

HASHTEXTCHAR [Variable]

The character separating two key hash keys. Default is ↑A.

HFGROWTHFACTOR [Variable]

The ratio of total slots to used slots when a hash file is created. Default is 3.

REHASHGAG [Variable]

Flags whether to print message when rehashing; initially off. Default is NIL.

SYSHASHFILE [Variable]

The current hash file. Default is NIL.

SYSHASHFILELST [Variable]

An alist of open hash files. Default is NIL.

## Limitations

The system currently is able to manipulate files on CORE, DSK, FLOPPY and over the network, via leaf servers. Hash files can be used with NS servers only if they support random access files.

Due to the pointer size, only hash files of less than 6 million initial entries can be created, though these can grow to 14 million entries before automatic rehashing exceeds the pointer limit. The total file length is limited to 16 million bytes. No range checking is done for these limits.

Two-key hash files operate on pnames only, without regard to packages.

# HASH-FILE

Hash-File is similar to but not compatible with the library module, Hash. Hash-File is modeled after the Common Lisp hash table facility, and Hash was modeled after the Interlisp hash array facility.

Hash files, like hash tables, are objects which efficiently map from a given Lisp object, called the *key*, to another Lisp object, called the *value*. Hash tables store this mapping in memory, while hash files store the mapping in a specially formatted file. Hash files are generally slower to access than hash tables, but they do not absorb memory and they are persistent over Lisp images. Hash files are recommended for large databases which do not change very often.

Since hash files are not stored in memory, hashing for EQ or EQL keys does not make sense. Memory references written to file in one session will probably not be valid in another. For this reason, the default hashing is for EQUAL keys, and then only those which can be dependably printed and read.

All of the code for Hash-File is in a package called Hash-File. Througout this document Lisp symbols will be printed as though in a package which uses the packages Hash-File and Lisp.

## Requirements

Hash files must reside on a random-access device (not a TCP/IP file server).

## Installation

Load HASH-FILE.DFASL from the Library.

## Functions

Hash-File has functions to create a new hash file, to open and close existing hash files, and to store and retrieve data in hash files.

### Creating a Hash File

(make-hash-file *file-name size &key . keys*) [Function]

Creates and returns a new hash file in *file-name* opened for input and output. *Size* indicates the table size and should be an integer somewhat larger than the maximum number of keys under which you expect to store values in this hash file. (The hash file will grow as required, so this number need not be accurate. See the section , "Rehashing," below.) The keyword arguments are explained as this document progresses.

## Opening and Closing Hash Files

(open-hash-file *file-name &key :direction . other-keys*) [Function]

Opens an existing hash file and returns it. The *:direction* argument must be one of `:input` or `:io`. If opened for `:input` then storing values in the hash file will be disallowed. The default for *:direction* is `:input`. Other key arguments are the same as for make-hash-file and are explained as this document progresses.

(close-hash-file *hash-file*) [Function]

Closes the file for *hash-file*, ensuring that all data has been saved. The backing file is always kept coherent; thus the only reason to close the *hash-file* is to ensure that the backing file is properly written to disk. All the functions mentioned in this document which operate on hash files will open the file when necessary; thus it is safe to call `close-hash-file` at almost any time.

## Storing and Retrieving Data

(get-hash-file *key hash-file &optional default*) [Function]

Retrieves the value stored under *key* in *hash-file*. Returns *default* if there is nothing stored under *key*. The default for *default* is `nil`. Also returns a second value which is true if something was found under *key* and false otherwise.

(get-hash-file *key hash-file*) [Setf place]

Values can be stores in a hash file with:

`(setf (get-hash-file` *key hash-file*`)` *new-value*`)`

Accordingly `incf`, `decf`, `push`, pop and any other macro that accepts generalized variables will work with `get-hash-file`.

(map-hash-file *function hash-file*) [Function]

For each entry in *hash-file*, *function* is called with the key and value stored.

Note: It is unsafe to change a hash file while mapping over it. The integrity of the file may be lost.

(rem-hash-file *key hash-file*) [Function]

Removes any entry for *key* in *hash-file*. Returns `t` if there was such an entry, `nil` otherwise.

## Other Functions

(copy-hash-file *hash-file file-name &optional new-size*) [Function]

Makes and returns a hash file in *file-name* with the same contents as *hash-file*. Much slower than `il:copyfile`, but performs garbage collection, often resulting in a smaller file.

(hash-file-count *hash-file*) [Function]

Returns the number of entries in *hash-file*.

(hash-file-p *object*) [Function]

Returns `t` if *object* is a hash file, `nil` otherwise.

`(hash-file-p` *object*`) ≡ (typep` *object* `'hash-file)`

## File Format

Hash-File uses a linked bucket implementation as illustrated in Figure 3.

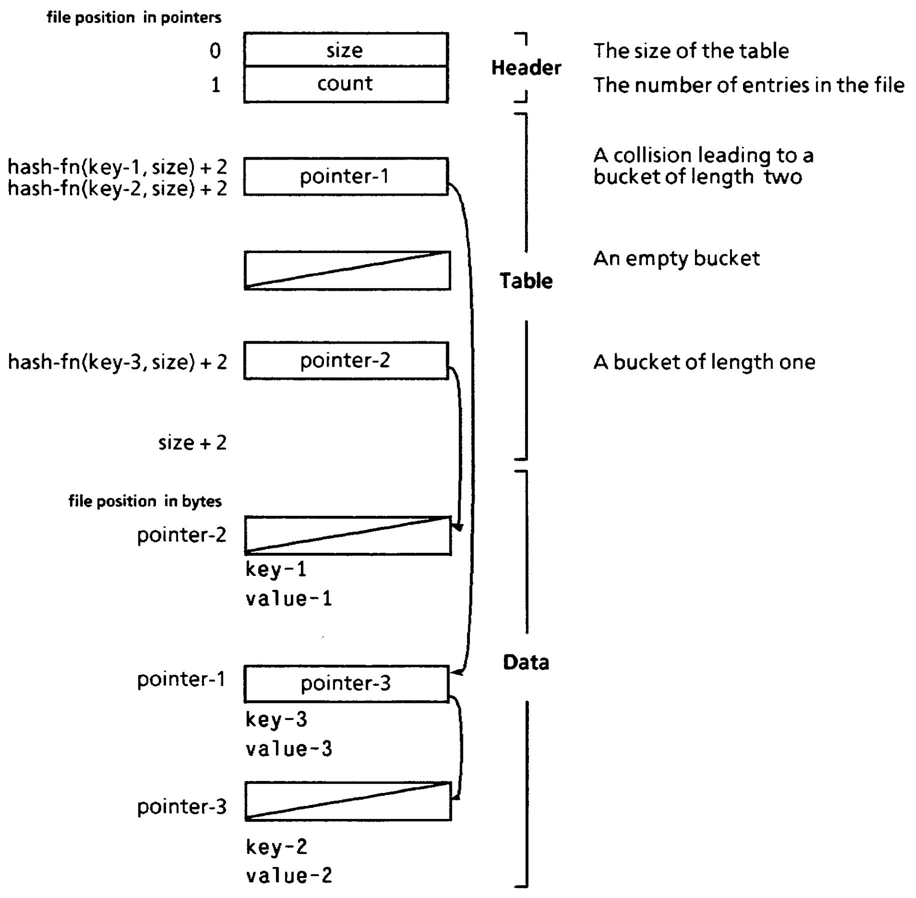


*Figure 3. Hash File Format*

Pointers are 32-bit integers written as four 8-bit bytes. There are two pointers of header (holding the size and count) followed by *size* pointers of table. Except for in the header and null pointers, all pointers are file-positions in bytes. Every such pointer points to the position on the file of the next pointer in the bucket. Immediately following the next pointer on the file are the printed representation of the key and value for the entry. New entries, including ones for old keys, are always added at the end of the file.

## Rehashing

When the number of keys with values in the file reaches a threshold, rehashing is performed to keep bucket lengths from getting too long. This threshold is expressed as a fraction of the table size.

rehash-threshold [Keyword argument]

Should be floating point number between zero and one. When the product of the table size and the rehash threshold of a hash file is greater than its `hash-file-count` then the hash file is automatically rehashed. The default for this keyword argument is the value of the special variable `hash-file::*rehash-threshold*` whose global binding is by default 0.875.

Rehashing is accomplished by having `copy-hash-file` make a new hash file with a larger size in a new version of the file. The new hash file structure is then smashed into the old one so that pointers to the old one are still valid.

rehash-size [Keyword argument]

Should be floating point number larger than one. The next prime larger than the product of this and the old table size is used to as the size for the new table. The default for this keyword argument is the value of the special variable `hash-file::*rehash-size*` whose global binding is by default 2.0.

hash-file::*delete-old-version-on-rehash* [Special variable]

If true, when rehashing generates a new version of the backing file the old version will be automatically deleted. The default top-level value for this variable is `nil`.

Rehashing is very expensive. Thus, when possible, you should attempt to make good estimates for the `size` argument to `make-hash-file`.

## Programmer's Interface

There may be applications in which you want to store things in hash files but which could not be printed and read by the functions `print` and `read`. The following hooks are provided for this purpose.

value-read-fn [Keyword argument]

Called by `get-hash-file` with one argument of a stream to read a value. The file position will be set to the same position as it was when this value was written. Default is `hash-file::default-read-fn` which binds `*package*` to the XCL package and `*readtable*` to the XCL readtable before calling `read`.

value-print-fn [Keyword argument]

Called by the setf method for `get-hash-file` with the object to be stored and the stream to print it on. The file position of the stream will be at the end of the file and there are no limitations as to how much can be printed. Default is `hash-file::default-print-fn` which binds `*package*` to the XCL package, `*readtable*` to the XCL readtable and `*print-base*` to 10 before calling `print`.

Example: A hash file with circular values.

```
(defun print-circular-object (object stream)
   (let ((*print-circle* t))
      (hash-file::default-print-fn object stream)))

(setq hash-file-with-circular-values
   (make-hash-file "{core}foo" 10
                                                 :value-print-fn
#'print-circular-object))

(setq l (list "foo"))
(setf (cdr l) l) ⇒ #1= ("foo" . #1#)

(setf (get-hash-file "bar" hash-file-with-circular-values)
l)

(get-hash-file "bar" hash-file-with-circular-values)
   ⇒ #1= ("foo" . #1#)

(eq * l) ⇒ nil
```

key-read-fn [Keyword argument]

Called by `get-hash-file` with one argument of a stream to read a key. The file postion will be set to the same position as it was when this key was written. Default is `hash-file::default-read-fn`, described above.

key-print-fn [Keyword argument]

Called by the setf method for `get-hash-file` with the object to be stored and the stream to print it on. The file position of the stream will be at the end of the file and there are no limitations as to how much can be printed. Default is `hash-file::default-print-fn`, described above.

Note: The value reader is called immediately after the key reader. Thus, the key reader must be sure to read all that the key printer printed so that the file position is appropriate for the value reader. However, the value reader is free to not read all that the value printer printed.

You might now think that you could make a hash file whose keys were circular by simply specifying our circular reader and printer for the key print and read functions, but this would not be sufficient. You also need the following hooks:

key-compare-fn [Keyword argument]

Called when searching a bucket to determine whether the correct key/value pair has been reached yet. Default is `equal`.

key-hash-fn [Keyword argument]

Called with a key and a range. Should return an integer between zero and range-1 with the following property:

key-hash-fn(x) = key-hash-fn(y) iff key-compare-fn(x,y)

The default `key-hash-fn` is `hash-file::hash-object` which works on symbols, strings, lists, bit-vectors, pathnames, characters and numbers. (Any object whose printed representation can be dependably read in as an object equal to the original.)

Note: This function will work on circular lists, as it only proceeds a fixed depth down a structure. Thus to hash on circular keys you also need to provide a key comparer which is able to compare circular keys, as most defintions of equal are not.

## Performance

A linked bucket implementation generally gives shorter bucket lengths, but uses more file space. The effects of this upon performance are difficult to judge.

The following table shows the distribution of bucket lengths in a Where-Is hash file containing 27,157 entries with a table size of 50,021.

| length | number of buckets this length |
|---|---|
| 0 | 29,279 (empty buckets) |
| 1 | 15,461 |
| 2 | 4334 |
| 3 | 794 |
| 4 | 125 |
| 5 | 23 |
| 6 | 4 |
| 7 | 1 |

This information was gathered by the function `hash-file::histogram`.

[This page intentionally left blank]

# HRULE

HRule is a module that lets you create horizontal rules (solid horizontal lines of various thicknesses) in a TEdit document. Rules are often used to set off titles and page headings from regular text, and to create decorative effects.

## Requirements

IMAGEOBJ
EDITBITMAP
TEDIT

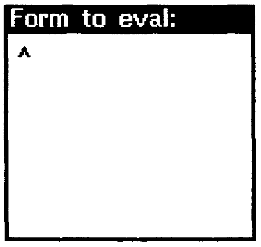
## Installation

Load HRULE.LCOM and the required .LCOM modules from the library.

## Creating Horizontal Rules

You specify a rule's thickness in decimal fractions of a printer's point (1/72 of an inch).

To create a horizontal rule, place the caret at the point in your document where you want the rule to begin, then type control-O. This will bring up a small window titled "Form to Eval" that contains a blinking caret. Type (HRULE.CREATE *N*) after the caret, with *N* indicating the thickness of the rule.



For example, to create a 4-point rule you would type (HRULE.CREATE 4); to create a 2½-point rule you would type (HRULE.CREATE 2.5). Then press the carriage return. The window will close, and a rule of the specified size will be created, extending from the TEdit caret to the right margin of the paragraph.

Note: This means that nothing can appear to the right of a rule on the same line.

So, for example if you type the following paragraph

This is an example of a paragraph that is about to have a horizontal rule inserted in it, to show what happens.

and insert a 2$\frac{1}{2}$-point rule after the word "rule," you end up with

This is an example of a paragraph that is about to have a horizontal rule
inserted in it, to show what happens.

Like other image objects in TEdit, a rule is a single character that can be deleted, moved, and copied like any other character.

You can use the TEdit Paragraph Looks menu to change the width of a rule if you don't want it to extend to the normal right margin of your document.

## Stacking Several Rules in a Single Object

Sometimes, you will want to stack several rules atop one another, with space between them. This can be used to achieve effects like

and

To create built-up rules of this type, follow the same procedure as above, but provide a list of rule widths and spacings in place of the single rule width. The first example above was created using the form (HRULE.CREATE '(.5 .5 .5)), and the second example was created using the form (HRULE.CREATE '(3 1 1 1 3)). The first number in the list is the thickness of the topmost rule, the next number is the space below it, the third number is the next rule, and so on.

## Limitations

A rule can be, theoretically, infinitely small or infinitely large. For most documents, however, you will probably want to create rules that are between half a point and six points thick. On printers, you can't usually tell the difference between rules that are less than $\frac{1}{2}$ point apart in thickness.

## Examples

Shown in Figure 4 are some examples of horizontal rules. In addition, you might want to look at the rules in this document, which were all created with HRule.

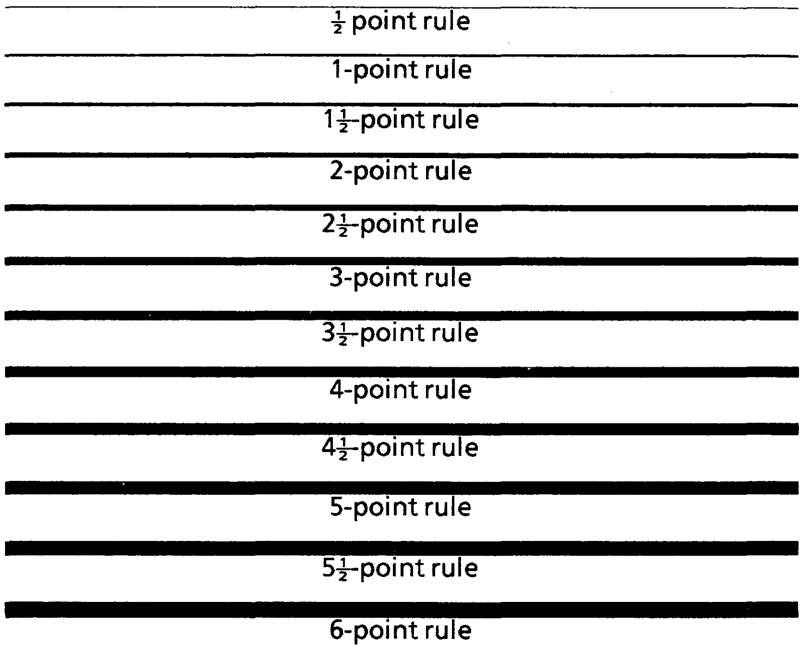


*Figure 4. Horizontal rules*

Shown in Figure 5 are some examples of built-up rules, along with what you would type to create them:

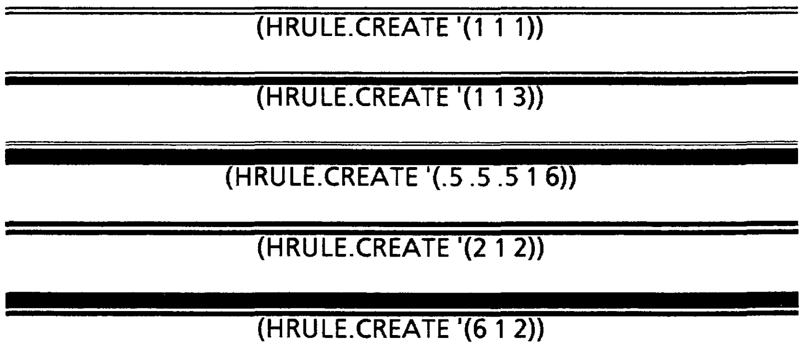


*Figure 5. Built-up rules*

[This page intentionally left blank]

# KERMIT AND MODEM

Kermit and Modem are utilities for transferring files between computers using ordinary RS232 and modem connections.

The file KERMIT.LCOM contains both the Kermit and Modem protocols. Once loaded, it provides a means of transferring files between a Xerox workstation and any other computer that supports either Kermit or Modem, and to which Lisp is able to open a Chat connection.

Of these two file transfer protocols, Kermit is preferred. Modem is much less flexible than Kermit, and cannot be used on RS232 connections requiring parity or flow control. Modem was developed primarily to support file transfers to and from microcomputers running the CP/M operating system. Modem implementations are available for Tops-20, VAX/Unix, and VAX/VMS. Kermit, on the other hand, was designed for file transfers between computers of many types, and there exist implementations of the Kermit protocol on machines ranging in size from eight-bit microcomputers to large IBM mainframes.

For a detailed discussion and tutorial on Kermit, see *Kermit: A File Transfer Protocol* by Frank Da Cruz, Digital Press, 1987.

## Requirements

The machine must run Kermit or Modem, and you need the means of reaching it, typically via Chat over an RS232 or a network connection.

You also need the following .LCOM files in order to run this module successfully:

KERMIT, KERMITMENU

as well as CHAT,

and either the RS232C or TCP-IP protocols, or the built-in NS or PUP protocols.

## Installation

Load KERMIT.LCOM and the required .LCOM modules from the library.

## Establishing a Connection

The first step in using Kermit or Modem is to establish a Chat connection with a desired host. You may use any sort of Chat connection (e.g., NS, TCP, PUP, or RS232). See the Chat module in this manual.

If you are using an RS232 connection, and plan to transfer files with the Modem protocol, do not establish a connection that requires parity to be used; establish the connection with eight bits per character and no parity (see the RS232 module in this manual). Disable flow control (XOn/XOff) when using Modem.

When you have established a Chat connection to a remote host, log in (if necessary) and start the remote host's Kermit or Modem program. The details of running these programs differ slightly between implementations; you should obtain documentation specific to the version of Kermit or Modem running on the remote host.

## Kermit

### Remote Kermit in Server Mode

Most mainframe implementations of Kermit have a server mode. This mode causes the remote Kermit to listen for either send or receive requests without your having to type additional commands to the remote Kermit. If the version of Kermit you are using on the remote host does support server mode, give the server mode command to place the program in this mode. In most implementations of Kermit, server mode is entered by your typing SERVER to the Kermit prompt:

```
Kermit>SERVER
```

### Remote Kermit Not in Server Mode

If the remote Kermit does not support server mode, you must issue individual send and receive requests for each file you transfer. To send a file to a remote Kermit, issue the RECEIVE command to the remote Kermit. To receive a file from a remote Kermit, issue the SEND command to the remote Kermit. In most cases, these commands are followed by the name of the file to be sent or received.

For example:

Kermit> RECEIVE *FILENAME*

or

Kermit> SEND *FILENAME*

If you are transferring files between two Xerox workstations connected by an RS232 connection, call (CHAT 'RS232) on each machine to establish the connection. Currently, Lisp Kermit does not support a server mode, so you must issue a receive request on one machine, followed by a send request on the other (see below).

After you have started the remote Kermit program, you need to start the local Lisp Kermit program. Lisp provides both functional and interactive interfaces for Kermit (and Modem).

### Local Kermit

To start the local side of the Kermit file transfer, use the KERMIT.SEND or KERMIT.RECEIVE functions:

(KERMIT.SEND *LOCALFILE REMOTEFILE WINDOW TYPE*) [Function]

*LOCALFILE* is the name of the file being sent to the remote Kermit.

*REMOTEFILE* is the name under which the file should be stored remotely. In most implementations of Kermit, this name overrides any name you specified in the remote receive command.

*WINDOW* is a pointer to the Chat window over which the transfer will take place. If *WINDOW* is NIL, the value of CHATWINDOW (the first Chat window to be opened) will be used in its place.

*TYPE* is the type of the file. It should be set to either TEXT or BINARY.

(KERMIT.RECEIVE *REMOTEFILE LOCALFILE WINDOW TYPE*) [Function]

*LOCALFILE* is the local name of the file to be received from the remote Kermit.

*REMOTEFILE* is the name of the file on the remote machine.

*WINDOW* is a pointer to the Chat window over which the transfer will take place. If *WINDOW* is NIL, the value of CHATWINDOW (the first Chat window to be opened) will be used in its place.

*TYPE* is the type of the file. It should be set to either TEXT or BINARY.

While the file transfer is in progress, the associated Chat window will be blank, and cumulative packet counts and other messages will be displayed in a one-line prompt window above the Chat window.

## Modem

To transfer files with the Modem protocol, you must run the Modem program on the remote machine. Modem does not support a server mode. Typically, you run the program once per file transferred, with instructions in the command line to indicate whether the file is being sent or received. There are a number of versions of the Modem protocol. On some systems, you run the program called Modem; on other systems, the program is called UModem or XModem.

On Unix, for instance, to send a text file to a Xerox workstation, you would type:

```
%XMODEM -ST FILENAME
```

On Tops-20, you would type:

```
@MODEM SA FILENAME
```

Note: % and @ are host system prompts.

As with Kermit, after you have started the remote side of the file transfer, you must start the local (Lisp) side. To do this, use either of the functions MODEM.SEND or MODEM.RECEIVE:

(MODEM.SEND *LOCALFILE WINDOW TYPE EOLCONVENTION*) [Function]

*LOCALFILE* is the name of the file to send to the remote Modem program.

*WINDOW* is the Chat window over which the transfer will take place.

*TYPE* is the file type, either TEXT or BINARY.

*EOLCONVENTION* is the end-of-line convention used by the operating system on which the remote Modem program is running. *EOLCONVENTION* should be one of CR, LF, or CRLF. Typically, Unix and VMS require LF, Tops-20 requires CRLF, and other Xerox machines require CR.

(MODEM.RECEIVE *LOCALFILE WINDOW TYPE EOLCONVENTION*) [Function]

*LOCALFILE* is the name of the file to receive from the remote Modem program.

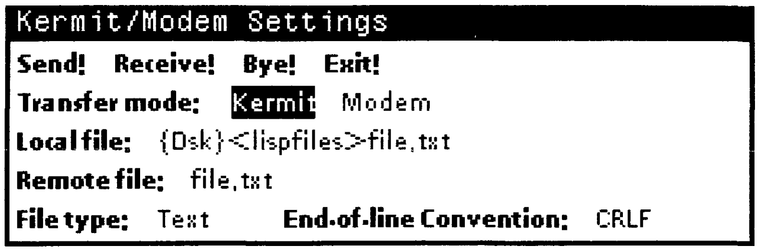
*WINDOW* is the Chat window over which the transfer will take place.

*TYPE* is the file type, either TEXT or BINARY.

*EOLCONVENTION* is the end-of-line convention used by the operating system on which the remote Modem program is running (see above).

## Interactive File Transfers With Kermit or Modem

A more convenient user interface for Kermit and Modem is available via the module KERMITMENU.LCOM. It provides a menu-oriented interface for issuing Kermit or Modem commands. To obtain the menu interface, press the middle mouse button in a live Chat window. The standard middle-button Chat menu will contain an entry labeled "Kermit" near its top. If you select this entry, a Kermit menu will appear at the top of the associated Chat window:



Kermit/Modem Settings

**Send! Receive! Bye! Exit!**

**Transfer mode:** Kermit Modem

**Local file:** {Dsk}<lispfiles>file.txt

**Remote file:** file.txt

**File type:** Text **End-of-line Convention:** CRLF

The entries on the top line of the menu are action commands:

| | |
|---|---|
| SEND | Starts sending a file to the remote Kermit or Modem program. The remote program must be prepared to receive the file. |
| RECEIVE | Starts receiving a file from the remote Kermit or Modem program. The remote program must already be attempting to send the file. |
| BYE | Closes (severs) the connection. |
| EXIT | Closes the window containing the menu, but does not close the connection. |
| TRANSFER MODE | This entry controls whether files are transferred using Kermit or Modem. You may set the state of this entry by selecting either of the Kermit or Modem labels with the mouse. The current transfer mode choice is displayed inverted in the menu. |
| LOCAL FILE | This entry holds the name of the local file being sent or received. You may set the contents of this field by selecting the LOCAL FILE label and typing the name. |
| REMOTE FILE | This entry holds the name of the remote file being stored or retrieved. You may set the contents of this field by selecting the REMOTE FILE label and typing the name. The Modem protocol does not use the contents of this field. |
| FILE TYPE | This field controls whether files are sent in binary or text (ASCII) mode. To set this field, select the FILE TYPE label and choose an entry from the menu that appears. |
| END-OF-LINE CONVENTION | This field sets the end-of-line convention being used by the remote Modem program (it is not used when files are transferred in Binary mode or with the Kermit protocol). The contents of this field must match the conventions of the operating system on which the remote Modem program is running. To set this field, select the END-OF-LINE CONVENTION label, and choose an entry from the menu that appears. |

## Limitations

Transfer files between two Xerox machines using the Kermit protocol.

Modem cannot be used on RS232 connections requiring parity or flow control.

[This page intentionally left blank]

# KEYBOARDEDITOR

KeyboardEditor is intended for use with the VirtualKeyboards module. You should read that module's documentation before reading this. The KeyboardEditor module lets you create new virtual keyboards and change existing ones to suit your needs.

## Requirements

VIRTUALKEYBOARDS

## Installation

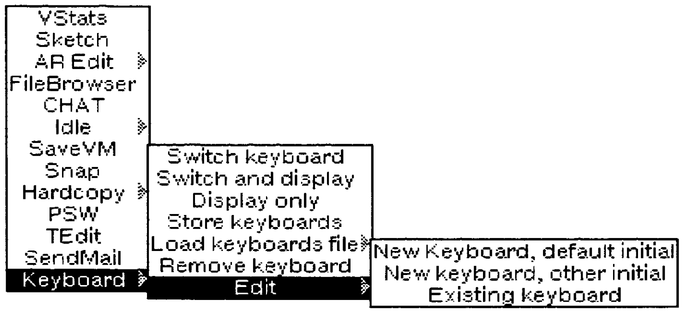
Load KEYBOARDEDITOR.LCOM and VIRTUALKEYBOARDS.LCOM from the library.

## User Interface

Loading KeyboardEditor adds EDIT to the Virtual Keyboard submenu on the background menu.

### Background Menu

The keyboard editor is used to modify and create virtual keyboards. You can call it by selecting EDIT from the main KeyboardEditor/VirtualKeyboards menu and sliding the cursor to the right to bring up the editor menu. You can also simply select EDIT, which gives you the same options as NEW KEYBOARD, DEFAULT INITIAL.

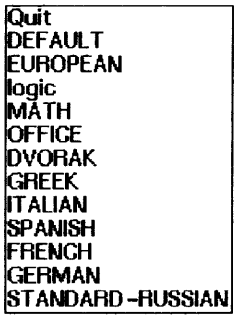


#### Creating a New Keyboard From a Copy of the Default Keyboard

Choose NEW KEYBOARD, DEFAULT INITIAL to create a keyboard from a copy of the default keyboard (which initially has the same key assignments as the 1108 keyboard). The system will prompt you for a name for the new keyboard, then call the editor with a copy of the default keyboard as the initial keyboard. The key assignments that are not changed during the editing session will remain as they are in the default keyboard.

### Creating a New Keyboard From a Copy of Any Known Keyboard

To create a new keyboard from a copy of a known keyboard other than the default keyboard, select NEW KEYBOARD, OTHER INITIAL from the Edit submenu. You will be prompted for a name for the new keyboard. The system will then display a menu of the known keyboards to enable you to choose one of them as the initial keyboard.



### Changing an Existing Keyboard

You can change an existing keyboard by selecting EXISTING KEYBOARD from the Edit submenu. Like the NEW KEYBOARD, OTHER INITIAL command, this brings up a menu of known keyboards from which you can choose a keyboard for editing. However, you will not be prompted for a keyboard name first, because you are editing the actual keyboard rather than using it as a base for a new keyboard.

## Calling the Keyboard Editor From Lisp

The editor can also be called using the function

(EDITKEYBOARD *KEYBOARD INITIALKEYBOARD*) [Function]

where *KEYBOARD* is either a virtual keyboard (i.e., a list) or the name of a virtual keyboard. If *KEYBOARD* is a virtual keyboard or the name of a known keyboard (a keyboard that was defined before), the editing will be done on that keyboard and the second argument will be ignored.

If *KEYBOARD* is a new name, the editing will be done on a copy of *INITIALKEYBOARD*, with *KEYBOARD* as its new name. If *INITIALKEYBOARD* is NIL, the default keyboard will be used as a base keyboard.

Examples:

To create a totally new virtual keyboard, call (EDITKEYBOARD *NEWNAME*).

To create a new keyboard that is similar to a keyboard with the name K1, call (EDITKEYBOARD *NEWNAME* 'K1)

To modify a keyboard with the name GREEK, call (EDITKEYBOARD 'GREEK).

## Using the Keyboard Editor

There are four different keyboard editor menus, three of them displayed at any given time. After you call the editor, you will see the command menu at the top, the character menu in the middle, and the keys menu at the bottom.

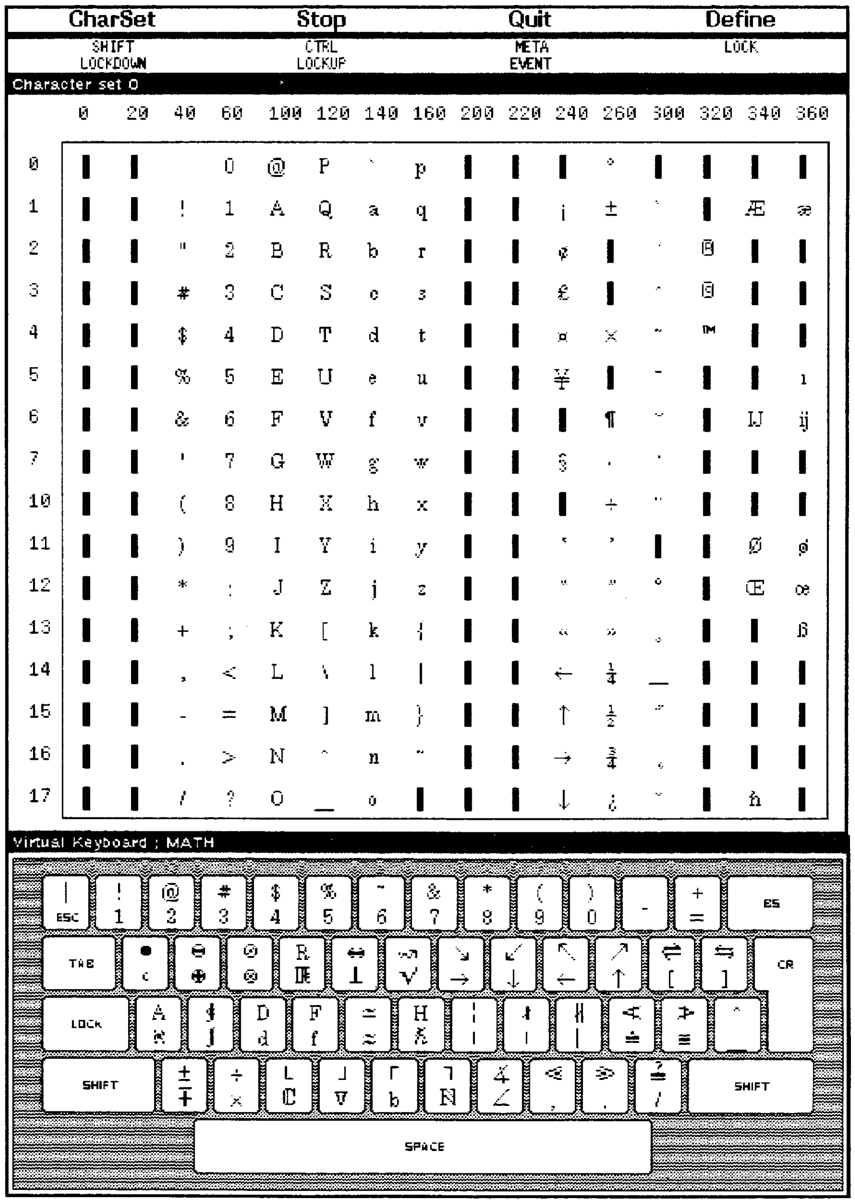
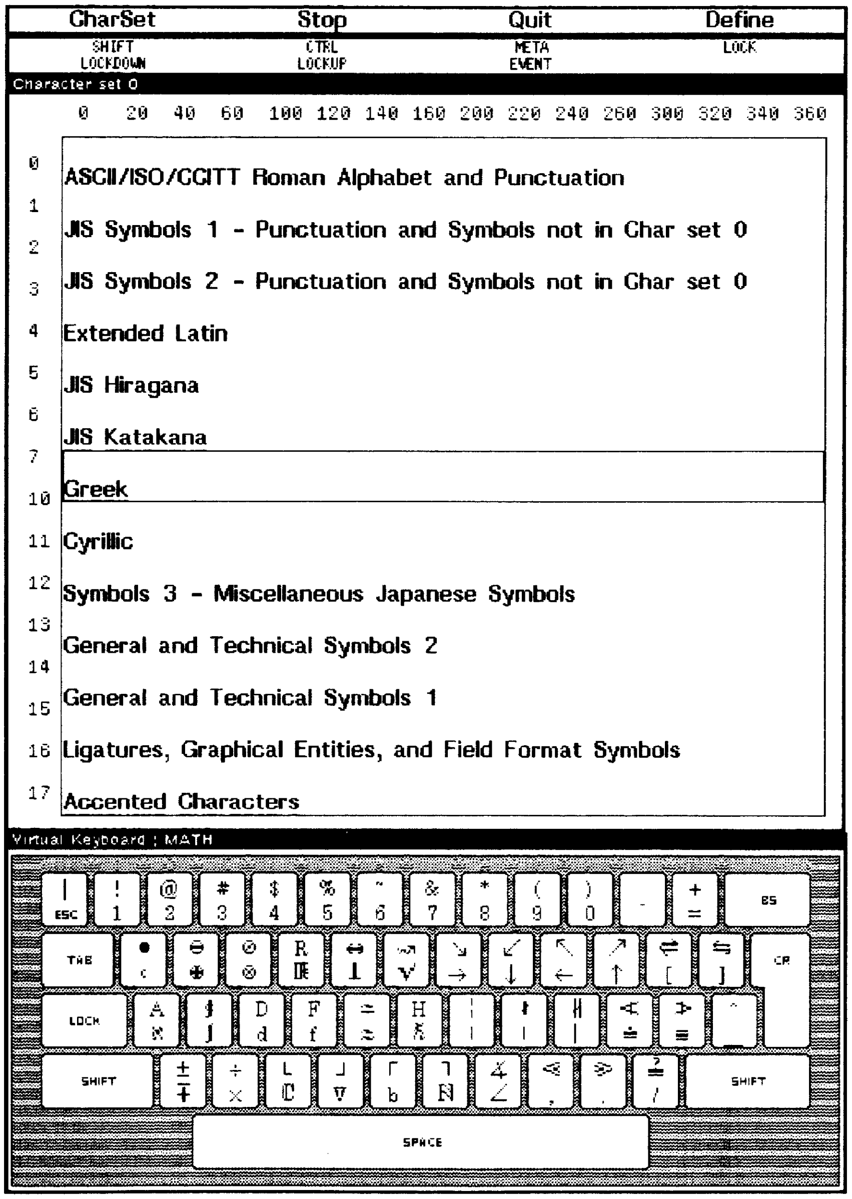


*Figure 6. Character Display*

The character menu is a 16-by-16-character display of the 256 characters available in the current character set. The set that is displayed when you enter the editor is character set 0, which includes all of the ASCII characters plus many other symbols. See Figure 6. If you need characters from other character sets, you have to select Char Set from the command menu. A new menu will pop up that contains numbers from 0 to 377 octal. This is the character set menu, and it lets you switch the character menu to display characters from other sets. Most of the character set numbers are not currently implemented. The most useful ones are shown in Figure 7.



```
   0  20  40  60  100 120 140 160 200 220 240 260 300 320 340 360

0    ASCII/ISO/CCITT Roman Alphabet and Punctuation
1
2    JIS Symbols 1 - Punctuation and Symbols not in Char set 0
3    JIS Symbols 2 - Punctuation and Symbols not in Char set 0
4    Extended Latin
5    JIS Hiragana
6    JIS Katakana
7
10   Greek
11   Cyrillic
12   Symbols 3 - Miscellaneous Japanese Symbols
13
14   General and Technical Symbols 2
15   General and Technical Symbols 1
16   Ligatures, Graphical Entities, and Field Format Symbols
17   Accented Characters
```

*Figure 7. Character Sets*

The keys menu lets you make a key the current key by selecting it. A selected key is marked by a black frame. To make a shifted key the current key, shift-select the key (hold the shift key down and click on the icon with the left button); it will be marked by inverted shift keys in addition to the black frame.

The basic operation of editing is assigning a character to a key. You can only assign character keys; keys other than character keys will retain their current definitions. You assign a character to a key by selecting the key from the keys menu, then selecting the character from the character menu. If the character is to be assigned to the shifted key, select the shifted key as the current key.

A second type of editing operation is to change the LOCKSHIFT state of a key. Each key either has or does not have a LOCKSHIFT property. If a key has a LOCKSHIFT property and the shift lock key of the keyboard is down, typing the key on your workstation keyboard will send the shifted character of the key, regardless of the state of the shift keys. The same rule applies to a virtual displayed keyboard; if the LOCK item is inverted and the key has a LOCKSHIFT property, selecting a key will send the shifted character to the current input stream.

If a key has the LOCKSHIFT property, the lock key will be inverted in the keys menu. To change the LOCKSHIFT property of a key, first make the shifted key the current key. You then set or unset the LOCKSHIFT property by selecting the lock key from the keys menu.

If you are creating a new keyboard and you are satisfied with the key assignments, select Define from the command menu. This will add the newly created keyboard to the list of known keyboards (it will thus appear on future menus). Selecting QUIT will exit after modifying the virtual keyboard, and selecting Stop will exit without modifying the keyboard. In both cases the new keyboard will be returned to the caller of EDITKEYBOARD function (above).

## Creating New Keyboard Configurations

KEYBOARDCONFIGURATION [Record]

Describes a physical keyboard: its layout, the key numbers that are used with KEYACTION. It also describes each key: its default meaning, its default label, whether you can change the key's meaning with the keyboard editor.

A configuration consists of a number of parts:

CONFIGURATIONNAME [Record field]

The name of this configuration.

For example, KeyboardEditor comes with configurations named DANDELION (1108), DORADO (1132), DOVE (1186), and FULL-IBMPC.

KEYSIDLIST [Record field]

A list of the IDs you will use for the keys in the rest of the configuration; i.e., your names for the keys. For simplicity, these are usually numbers starting beyond 100 (to avoid overlapping the true range of key numbers).

KEYREGIONS [Record field]

An alist of key IDs and the regions they occupy in the keyboard's image when it is displayed. For example, the alphabetic keys in the DANDELION keyboard are 29 screen points wide and 33 high.

DEFAULTASSIGNMENT [Record field]

An alist of key IDs and their default KEYACTIONs (see *IRM*).

KEYNAMESMAPPING [Record field]

An alist of key names to key IDs. The key names should be mnemonic, and should distinguish relevant differences; e.g., the 7 on the 1186's numeric keypad is named NUMERIC7, while the 7 key in the main keyboard cluster is named 7.

MACHINETYPE [Record field]

The kind of machine for which this configuration is intended.

For example, the FULL-IBMPC configuration is meant to be used with a DAYBREAK keyboard, so its MACHINETYPE is DAYBREAK.

KEYLABELS [Record field]

An alist of key numbers to special labels. This is used to label keys such as the "Next" key, where the key assignment may not be a printable character.

KEYLABELSFONT [Record field]

The font you want to use for the key labels. The default value is Helvetica 5.

BACKGROUNDSHADE [Record field]

The shading for the non-key parts of the virtual keyboard's image. This defaults to a reasonable gray value.

KEYBOARDDISPLAYFONT [Record field]

The font used to display actual character assignments. This should probably be Classic 12, since it is the most complete font.

CHARLABELS [Record field]

An alist from character codes to names. Used to give symbolic names to characters such as ESCAPE, which don't otherwise print.

ACTUALKEYSMAPPING [Record field]

A function that takes one of your key IDs and returns a true key number, for use by KEYACTION.

Note: To create a new configuration, create an instance of the KEYBOARDCONFIGURATION record, using the field names shown above. Then add it to the list

VKBD.CONFIGURATIONS. You may then edit it using the configuration editor described below.

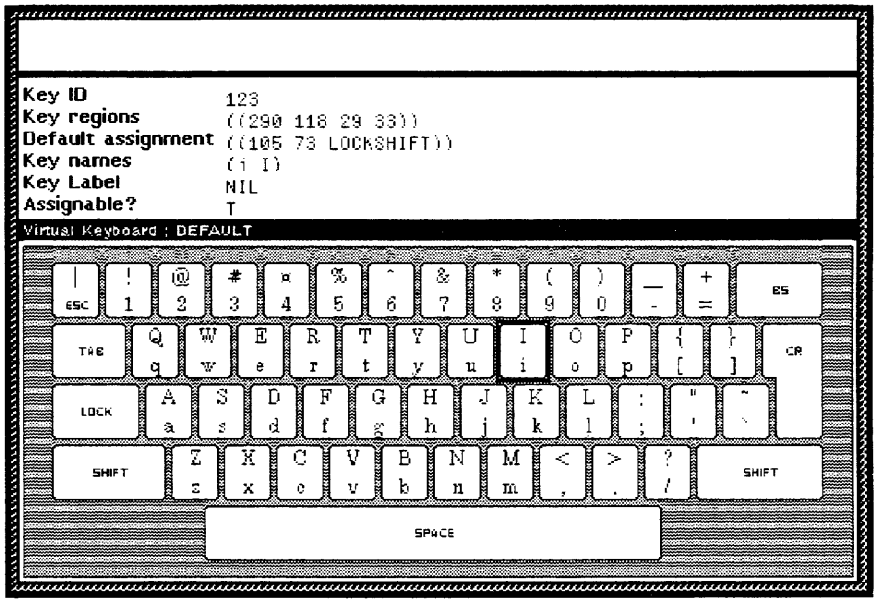
Note: You must save your own configurations. There is no user interface for saving them, nor any automatic scheme.

## Editing a Keyboard Configuration

Once you have created a KEYBOARDCONFIGURATION, you can make modest changes to it using the function:

(EDITCONFIGURATION *CONFIGNAME*) [Function]

where *CONFIGNAME* is the CONFIGURATIONNAME you have assigned to your new configuration. This will create a virtual keyboard display window with a menu on top of it as shown in Figure 8.



```
Key ID              123
Key regions         ((290 118 29 33))
Default assignment  ((105 73 LOCKSHIFT))
Key names           (i I)
Key Label           NIL
Assignable?         T
```

*Figure 8. Virtual keyboard display window*

Selecting a key with the mouse fills in the fields in the menu. The figure shows the 1108's configuration being edited, with the I key selected. To change one of the values, select the label at the left edge of the menu (e.g., ASSIGNABLE?). You will be prompted to edit the existing value using TTYIN.

The keyboard image is not automatically updated. To refresh it, select REDISPLAY in the right-button window menu.

When you have finished editing, simply close the keyboard window.

[This page intentionally left blank]

# MASTERSCOPE

MasterScope is an interactive program for analyzing and cross referencing user programs. It contains facilities for analyzing user functions to determine what other functions are called, how and where variables are bound, set, or referenced, and which functions use particular record declarations. MasterScope can analyze definitions directly from a file as well as in-memory definitions.

MasterScope maintains a data base of the results of the analyses it performs. Via a simple command language, you may interrogate the data base, call the editor on those expressions in functions that were analyzed which use variables or functions in a particular way, or display the tree structure of function calls among any set of functions.

MasterScope is interfaced with the editor and file manager so that when a function is edited or a new definition loaded in, MasterScope knows that it must reanalyze that function.

With the Medley release, MasterScope now understands Common Lisp **defun**, **defmacro**, and **defvar**.

## Requirements

MSANALYZE, MSPARSE, MSCOMMON, MS-PACKAGE

You may also want to make use of Browser, DataBaseFns, and SEdit or DEdit.

## Installation

Load MASTERSCOPE.DFASL and the other .DFASL files from the library.

## MasterScope Command Language

You communicate with MasterScope using an English-like command language, e.g., WHO CALLS PRINT. With these commands, you can direct that functions be analyzed, interrogate the MasterScope data base, and perform other operations. The commands deal with sets of functions, variables, etc., and relations between them (e.g., call, bind). Sets correspond to English nouns, relations correspond to verbs.

A set of atoms can be specified in a variety of ways, either explicitly, e.g., FUNCTIONS ON FIE specifies the atoms in (FILEFNSLST 'FIE), or implicitly, e.g., NOT CALLING Y, where the meaning must be determined in the context of the rest of the command. Such sets of atoms are the basic building blocks with which the command language deals.

MasterScope also deals with relations between sets.

For example, the relation CALL relates functions and other functions; the relations BIND and USE FREELY relate functions and variables. These relations get stored in the MasterScope data base when functions are analyzed. In addition, MasterScope "knows" about file manager conventions; CONTAIN relates files and various types of objects (functions, variables).

Sets and relations are used (along with a few additional words) to form sentence-like commands.

For example, the command WHO ON 'FOO USE 'X FREELY will print out the list of functions contained in the file FOO which use the variable X freely. The command EDIT WHERE ANY CALLS 'ERROR will call EDITF (see *IRM*) on those functions which have previously been analyzed that directly call ERROR, pointing at each successive expression where the call to ERROR actually occurs.

## MasterScope Commands

The normal mode of communication with MasterScope is via commands. These are sentences in the MasterScope command language which direct MasterScope to answer questions or perform various operations.

MasterScope commands are typed into the Executive window, preceded by a period (.) to distinguish them from other commands to the Exec. MasterScope keywords can be in any package, so MasterScope commands can be issued in any type of Exec. The commands may be typed uppercase or lowercase.

Note: Any MasterScope command may be followed by OUTPUT *FILENAME* to send output to the given file rather than the terminal, e.g. WHO CALLS WHO OUTPUT CROSSREF.

ANALYZE *SET* [MasterScope command]

Analyzes the functions in *SET* (and any functions called by them) and includes the information gathered in the data base. MasterScope will not reanalyze a function if it thinks it already has valid information about that function in its data base. You may use the command REANALYZE to force reanalysis.

Note that whenever a function is referred to in a command as a subject of one of the relations, it is automatically analyzed; you need not give an explicit ANALYZE command. Thus, WHO IN MYFNS CALLS FIE will automatically analyze the functions in MYFNS if they have not already been analyzed.

Note also that only EXPR definitions will be analyzed; that is, MasterScope will not analyze compiled code. If necessary, the definition will be DWIMIFYed before analysis. If there is no in-core definition for a function (either in the function definition cell or an EXPR property), MasterScope will attempt to read in the definition from a file. Files which have been explicitly mentioned previously in some command are searched first. If the definition cannot be found on any of those files, MasterScope looks among the files on FILELST for a definition. If a function is found in this manner, MasterScope will print a message "(reading from *FILENAME*)". If no definition can be found at all, MasterScope will print a message "*FN* can't be analyzed". If the function previously was known, the message "*FN* disappeared!" is printed.

REANALYZE *SET* [MasterScope command]

Causes MasterScope to reanalyze the functions in *SET* (and any functions called by them) even if it already has valid information in its data base. This would be necessary if you had disabled or subverted the file manager; e.g. performed PUTD's to change the definition of functions.

ERASE *SET* [MasterScope command]

Erases all information about the functions in *SET* from the data base. ERASE by itself clears the entire data base.

SHOW PATHS *PATHOPTIONS* [MasterScope command]

Displays a tree of function calls. This is described fully in "SHOW PATHS" below.

*SET RELATION SET* [MasterScope command]
*SET* IS *SET* [MasterScope command]
*SET* ARE *SET* [MasterScope command]

These commands have the same format as an English sentence with a subject (the first *SET*), a verb (*RELATION* or *IS* or *ARE*), and an object (the second *SET*). Any of the *SET*s within the command may be preceded by the question determiners WHICH or WHO (or just WHO alone).

For example, WHICH FUNCTIONS CALL X prints the list of functions that call the function X.

*RELATION* may be one of the relation words in present tense (CALL, BIND, TEST, SMASH, etc.) or used as a passive (e.g., WHO IS CALLED BY WHO). Other variants are allowed, e.g. WHO DOES X CALL, IS FOO CALLED BY FIE, etc.

The interpretation of the command depends on the number of question elements present:

If there is no question element, the command is treated as an assertion and MasterScope returns either T or NIL, depending on whether that assertion is true. Thus, ANY IN MYFNS CALL HELP will print T if any function in MYFNS call the function HELP, and NIL otherwise.

If there is one question element, MasterScope returns the list of items for which the assertion would be true.

For example,

```
MYFN BINDS WHO USED FREELY BY YOURFN
```

prints the list of variables bound by MYFN which are also used freely by YOURFN.

If there are two question elements, MasterScope will print a doubly indexed list:

```
_. WHO CALLS WHO IN /FNS
RECORDSTATEMENT --  /RPLNODE
RECORDECL1 --       /NCONC, /RPLACD, /RPLNODE
RECREDECLARE1 --    /PUTHASH
UNCLISPTRAN --      /PUTHASH, /RPLNODE2
RECORDWORD --       /RPLACA
RECORD1 --          /RPLACA, /SETTOPVAL
EDITREC --          /SETTOPVAL
```

EDIT WHERE *SET RELATION SET [- EDITCOMS]* [MasterScope command]

(WHERE may be omitted.) The first *SET* refers to a set of functions. The EDIT command calls the editor on each expression where the *RELATION* actually occurs.

For example, EDIT WHERE ANY CALL ERROR will call EDITF on each (analyzed) function which calls ERROR stopping within a TTY: at each call to ERROR. Currently you cannot EDIT WHERE a file which CONTAINS a datum, nor where one function CALLS another SOMEHOW.

*EDITCOMS*, if given, is a list of commands passed to EDITF to be performed at each expression.

For example,

```
EDIT WHERE ANY CALLS MYFN DIRECTLY - (SW 2 3) P
```

will switch the first and second arguments to MYFN in every call to MYFN and print the result. EDIT WHERE ANY ON MYFILE CALL ANY NOT @ GETD will call the editor on any expression involving a call to an undefined function.

Note that EDIT WHERE X SETS Y will point only at those expressions where Y is actually set, and will skip over places where Y is otherwise mentioned.

SHOW WHERE *SET RELATION SET* [MasterScope command]

Like the EDIT command except merely prints out the expressions without calling the editor.

EDIT *SET [- EDITCOMS]* [MasterScope command]

Calls EDITF on each function in *SET*. *EDITCOMS*, if given, will be passed as a list of editor commands to be Executed.

For example,

```
EDIT ANY CALLING FN1 - (R FN1 FN2)
```

will replace FN1 by FN2 in those functions that call FN1.

DESCRIBE *SET* [MasterScope command]

Prints the BIND, USE FREELY and CALL information about the functions in *SET*.

For example, the command DESCRIBE PRINTARGS might print out:

```
PRINTARGS[N,FLG]
   binds:       TEM,LST,X
   calls:       MSRECORDFILE,SPACES,PRIN1
   called by:   PRINTSENTENCE,MSHELP,CHECKER
```

This shows that PRINTARGS has two arguments, N and FLG; binds internally the variables TEM, LST and X; calls MSRECORDFILE, SPACES and PRIN1; and is called by PRINTSENTENCE, MSHELP, and CHECKER.

You can specify additional information to be included in the description. DESCRIBELST is a list each of whose elements is a list containing a descriptive string and a form. The form is evaluated (it can refer to the name of the funtion being described by the free variable FN). If it returns a non-NIL value, the description string is printed followed by the value. If the value is a list, its elements are printed with commas between them.

For example, the entry

```
("types:  " (GETRELATION FN '(USE TYPE) T)
```

would include a listing of the types used by each function.

CHECK *SET* [MasterScope command]

Checks for various anomalous conditions (mainly in the compiler declarations) for the files in *SET* (if *SET* is not given, FILELST is used).

For example, this command will warn about:

Variables which are bound but never referenced.

Functions in BLOCKS declarations which aren't on the file containing the declaration.

Functions declared as ENTRIES but not in the block.

Variables which may not need to be declared SPECVARS because they are not used freely below the places where they are bound.

etc.

FOR *VARIABLE SET I.S.TAIL* [MasterScope command]

This command provides a way of combining CLISP iterative statements with MasterScope. An iterative statement will be constructed in which *VARIABLE* is iteratively assigned to each element of *SET*, and then the iterative statement tail *I.S.TAIL* is Executed.

For example,

```
FOR X CALLED BY FOO WHEN CCODEP DO (PRINTOUT T X
,,,  (ARGLIST X) T)
```

will print out the name and argument list of all of the compiled functions which are called by FOO.

## MasterScope Relations

A relation is specified by one of the keywords below. Some of these "verbs" accept modifiers.

For example, USE, SET, SMASH and REFERENCE all may be modified by FREELY. The modifier may occur anywhere within the command. If there is more than one verb, any modifier between two verbs is assumed to modify the first one.

For example, in

```
USING ANY FREELY OR SETTING X,
```

FREELY modifies USING but not SETTING. The entire phrase is interpreted as the set of all functions which either use any variable freely or set the variable X, whether or not X is set freely. Verbs can occur in the present tense (e.g., USE, CALLS, BINDS, USES) or as present or past participles (e.g., CALLING, BOUND, TESTED). The relations (with their modifiers) recognized by MasterScope are:

CALL [MasterScope relation]

Function F1 calls F2 if the definition of F1 contains a form (F2 --). The CALL relation also includes any instance where a function uses a name as a function, as in

`(APPLY (QUOTE F2) --), (FUNCTION F2)`, etc.

(CALL and CALLS are equivalent.)

CALL SOMEHOW [MasterScope relation]

One function calls another SOMEHOW if there is some path from the first to the other. That is, if F1 calls F2, and F2 calls F3, then F1 CALLS F3 SOMEHOW.

This information is not stored directly in the data base; instead, MasterScope stores only information about direct function calls, and (re)computes the CALL SOMEHOW relation as necessary.

USE [MasterScope relation]

If unmodified, the relation USE denotes variable usage in any way; it is the union of the relations SET, SMASH, TEST, and REFERENCE.

SET [MasterScope relation]

A function SETs a variable if the function contains a form

`(SETQ var --), (SETQQ var --)`, etc.

SMASH [MasterScope relation]

A function SMASHes a variable if the function calls a destructive list operation (RPLACA, RPLACD, DREMOVE, SORT, etc.) on the value of that variable. MasterScope will also find instances where the operation is performed on a part of the value of the variable. For example, if a function contains a form (RPLACA (NTH X 3) T), it will be noted as SMASHing X.

If the function contains a sequence (SETQ Y X), (RPLACA Y T), then Y is noted as being SMASHed, but not X.

TEST [MasterScope relation]

A variable is TESTed by a function if its value is only distinguished between NIL and non-NIL.

For example, the form (COND ((AND X --) --)) tests the value of X.

REFERENCE [MasterScope relation]

This relation includes all variable usage except for SET.

Note: The verbs USE, SET, SMASH, TEST and REFERENCE may be modified by the words FREELY or LOCALLY. A variable is used FREELY if it is not bound in the function at the place of its use. It is used LOCALLY if the use occurs within a PROG or LAMBDA that binds the variable.

MasterScope also distinguishes between CALL DIRECTLY and CALL INDIRECTLY. A function is called directly if it occurs as CAR-of-form in a normal evaluation context. A function is called indirectly if its name appears in a context which does not imply its immediate evaluation, for example (SETQ Y (LIST (FUNCTION FOO) 3)). The distinction is whether or not the compiled code of the caller would contain a direct call to the callee.

Note that an occurrence of (FUNCTION FOO) as the functional argument to one of the built-in mapping functions which compile open is considered to be a direct call.

In addition, CALL FOR EFFECT (where the value of the function is not used) is distinguished from CALL FOR VALUE.

BIND [MasterScope relation]

The BIND relation between functions and variables includes both variables bound as function arguments and those bound in an internal PROG or LAMBDA expression.

USE AS A FIELD [MasterScope relation]

MasterScope notes all uses of record field names within FETCH, REPLACE or CREATE expressions.

FETCH [MasterScope relation]

Use of a field within a FETCH expression.

REPLACE [MasterScope relation]

Use of a record field name within a REPLACE or CREATE expression.

USE AS A RECORD [MasterScope relation]

MasterScope notes all uses of record names within CREATE or TYPE? expressions. Additionally, in (fetch (FOO FIE) of X), FOO is used as a record name.

CREATE [MasterScope relation]

Use of a record name within a CREATE expression.

USE AS A PROPERTY NAME [MasterScope relation]

MasterScope notes the property names used in expressions such as GETPROP, PUTPROP, GETLIS, etc., if the name is quoted; e.g. if a function contains a form (GETPROP X (QUOTE INTERP)), then that function USEs INTERP as a property name.

USE AS A CLISP WORD [MasterScope relation]

MasterScope notes all iterative statement operators and user defined CLISP words as being used as a CLISP word.

CONTAIN [MasterScope relation]

Files CONTAIN functions, records, and variables. This relation is not stored in the data base but is computed using the file manager.

DECLARE AS LOCALVAR [MasterScope relation]
DECLARE AS SPECVAR [MasterScope relation]

MasterScope notes internal calls to DECLARE from within functions.

ACCEPT [MasterScope relation]
SPECIFY [MasterScope relation]
KEYCALL [MasterScope relation]

MasterScope notes keyword arguments of Common Lisp functions when they are analyzed and when they are called.

FOO ACCEPTS :BAR is true if FOO is a Common Lisp function that accepts the keyword :BAR. FOO ACCEPTS &ALLOW-OTHER-KEYS is true if FOO has &ACCEPT-OTHER-KEYS in its lambda list.

FOO SPECIFIES :BAR is true if FOO is a function that calls any function with the keyword :BAR; the function in question must ACCEPT :BAR.

FOO KEYCALLS BAR is true if FOO is a function and calls BAR with one or more keywords it ACCEPTS.

FLET [MasterScope relation]
LABEL [MasterScope relation]
MACROLET [MasterScope relation]
LOCAL-DEFINE [MasterScope relation]

MasterScope tracks uses of Common Lisp local definition forms (it currently does not expand them while analyzing them, however).

FOO FLETS BAR is true of FOO is a function with a FLET defining BAR local to FOO.

LABELS and MACROLETS are similar. LOCAL-DECLARES is the union of FLETS, LABELS, and MACROLETS.

## Abbreviations

The following abbreviations are recognized:

FREE = FREELY
LOCAL = LOCALLY
PROP = PROPERTY
REF = REFERENCE

Also, the words A, AN and NAME (after AS) are "noise" words and may be omitted.

## MasterScope Templates

MasterScope uses templates (see "Effecting MasterScope Analysis" below) to decide which relations hold between functions and their arguments.

For example, the information that SORT SMASHes its first argument is contained in the template for SORT. MasterScope initially contains templates for most system functions which set variables, test their arguments, or perform destructive operations. You may change existing templates or insert new ones in MasterScope's tables via the SETTEMPLATE function (below).

MasterScope also constructs templates to handle Common Lisp functions with keyword arguments. These constructed templates are noticed by FILES? and can be saved if desired, or MasterScope can recreate them by analyzing the functions again.

## MasterScope Set Specifications

A set is a collection of things (functions, variables, etc.). A set is specified by a set phrase, consisting of a determiner (e.g., ANY, WHICH, WHO) followed by a type (e.g., FUNCTIONS, VARIABLES) followed by a specification (e.g., IN MYFNS). The determiner, type and specification may be used alone or in combination.

For example,

```
ANY FUNCTIONS IN MYFNS,
VARIABLES IN GLOBALVARS, and
WHO
```

are all acceptable set phrases.

Note: Sets may also be specified with relative clauses introduced by the word THAT, e.g. THE FUNCTIONS THAT BIND 'X.

*'ATOM* [MasterScope set specification]

The simplest way to specify a set consisting of a single thing is by the name of that thing.

For example, in the command WHO CALLS 'ERROR, the function ERROR is referred to by its name. Although the ' (apostrophe) can be left out, to resolve possible ambiguities names should usually be quoted; e.g., WHO CALLS 'CALLS will return the list of functions which call the function CALLS.

'*LIST* [MasterScope set specification]

Sets consisting of several atoms may be specified by naming the atoms.

For example, the command WHO USES '(A B) returns the list of functions that use the variables A or B.

IN *EXPRESSION* [MasterScope set specification]

The form *EXPRESSION* is evaluated, and its value is treated as a list of the elements of a set.

For example, IN GLOBALVARS specifies the list of variables in the value of the variable GLOBALVARS.

@ *PREDICATE* [MasterScope set specification]

A set may also be specified by giving a predicate which the elements of that set must satisfy. *PREDICATE* is either a function name, a LAMBDA expression, or an expression in terms of the variable X. The specification @ *PREDICATE* represents all atoms for which the value of *PREDICATE* is non-NIL.

For example, @ EXPRP specifies all those atoms which have EXPR definitions; @ (STRPOSL X CLISPCHARRAY) specifies those atoms which contain CLISP characters. The universe to be searched is either determined by the context within the command (e.g., in WHO IN FOOFNS CALLS ANY NOT @ GETD, the predicate is only applied to functions which are called by any functions in the list FOOFNS), or in the extreme case, the universe defaults to the entire set of things which have been noticed by MasterScope, as in the command WHO IS @ EXPRP.

LIKE *ATOM* [MasterScope set specification]

*ATOM* may contain ESCapes; it is used as a pattern to be matched, as in the editor.

For example, WHO LIKE /R$ IS CALLED BY ANY would find both /RPLACA and /RPLNODE.

(The ESC character prints out as a $; it is a wildcard for any number of characters.)

FIELDS OF *SET* [MasterScope set specification]

*SET* is a set of records. This denotes the field names of those records.

For example, the command WHO USES ANY FIELDS OF BRECORD returns the list of all functions which do a fetch or replace with any of the field names declared in the record declaration of BRECORD.

KNOWN [MasterScope set specification]

The set of all functions which have been analyzed.

For example, the command WHO IS KNOWN will print out the list of functions which have been analyzed.

THOSE [MasterScope set specification]

The set of things printed out by the last MasterScope question.

For example, following the command

```
WHO IS USED FREELY BY PARSE
```

you could ask WHO BINDS THOSE to find out where those variables are bound.

ON PATH *PATHOPTIONS* [MasterScope set specification]

Refers to the set of functions which would be printed by the command SHOW PATHS *PATHOPTIONS*.

For example,

```
IS FOO BOUND BY ANY ON PATH TO 'PARSE
```

tests whether FOO might be bound above the function PARSE (that is, whether FOO is bound in any function that is higher up in the calling tree than PARSE is) . SHOW PATHS is explained in detail below.

## Set Specifications by Relation

A set may also be specified by giving a relation its members must have with the members of another set:

*RELATION*ING *SET* [MasterScope set specification]

*RELATION*ING is used here generically to mean any of the relation words in the present participle form (possibly with a modifier), e.g., USING, SETTING, CALLING, BINDING. *RELATION*ING *SET* specifies the set of all objects which have that relation with some element of *SET*.

For example, CALLING X specifies the set of functions which call the function X; USING ANY IN FOOVARS FREELY specifies the set of functions which uses freely any variable in the value of FOOVARS.

*RELATION*ED BY *SET* [MasterScope set specification]
*RELATION*ED IN *SET* [MasterScope set specification]

This is similar to the *RELATION*ING construction.

For example, CALLED BY ANY IN FOOFNS represents the set of functions which are called by any element of FOOFNS; USED FREELY BY ANY CALLING ERROR is the set of variables which are used freely by any function which also calls the function ERROR.

## Set Specifications by Blocktypes

*BLOCKTYPE* OF *FUNCTIONS* [MasterScope set specification]
*BLOCKTYPE* ON *FILES* [MasterScope set specification]

These phrases allow you to ask about BLOCKS declarations on files (see *IRM*). *BLOCKTYPE* is one of LOCALVARS, SPECVARS, GLOBALVARS, ENTRIES, BLKFNS, BLKAPPLYFNS, or RETFNS.

*BLOCKTYPE* OF *FUNCTIONS* specifies the names which are declared to be *BLOCKTYPE* in any blocks declaration which contain any of *FUNCTIONS* (a "set" of functions). The "functions" in *FUNCTIONS* can either be block names or just functions in a block.

For example,

```
WHICH ENTRIES OF ANY CALLING 'Y BIND ANY
GLOBALVARS ON 'FOO.
```

*BLOCKTYPE* ON *FILES* specifies all names which are declared to be *BLOCKTYPE* on any of the given *FILES* (a "set" of files).

## Set Determiners

Set phrases may be preceded by a determiner, which is one of the words THE, ANY, WHO or WHICH. The question determiners (WHO and WHICH) are meaningful in only some of the commands, namely those that take the form of questions. ANY and WHO (or WHOM) can be used alone; they are wild-card elements, e.g., the command WHO USES ANY FREELY, will print out the names of all (known) functions which use any variable freely. If the determiner is omitted, ANY is assumed; e.g. the command WHO CALLS '(PRINT PRIN1 PRIN2) will print the list of functions which call any of PRINT, PRIN1, PRIN2. THE is also allowed, e.g. WHO USES THE RECORD FIELD FIELDX.

## Set Types

Any set phrase has a type; that is, a set may specify either functions, variables, files, record names, record field names or property names. The type may be determined by the context within the command (e.g., in CALLED BY ANY ON FOO, the set ANY ON FOO is interpreted as meaning the functions on FOO since only functions can be CALLED), or you may give the type explicitly (e.g., FUNCTIONS ON FIE).

The following types are recognized: FUNCTIONS, VARIABLES, FILES, PROPERTY NAMES, RECORDS, FIELDS, I.S.OPRS. Also, the abbreviations FNS, VARS, PROPNAMES or the singular forms FUNCTION, FN, VARIABLE, VAR, FILE, PROPNAME, RECORD, FIELD are recognized.

Note that most of these types correspond to built-in file manager types (see *IRM*).

The type is used by MasterScope in a variety of ways when interpreting the set phrase:

(1) Set types are used to disambiguate possible parsings.

For example, both commands

```
WHO SETS ANY BOUND IN X OR USED BY Y
WHO SETS ANY BOUND IN X OR CALLED BY Y
```

have the same general form. However, the first case is parsed as

```
WHO SETS ANY (BOUND BY X OR USED BY Y)
```

since both BOUND BY X and USED BY Y refer to variables; while the second case is parsed as

```
WHO SETS ANY BOUND IN (X OR CALLED BY Y),
```

since CALLED BY Y and X must refer to functions.

Note that parentheses may be used to group phrases.

(2) The type is used to determine the modifier for USE:

`FOO USES WHICH RECORDS` is equivalent to

```
FOO USES WHO AS A RECORD FIELD.
```

(3) The interpretation of CONTAIN depends on the type of its object: the command

```
WHAT FUNCTIONS ARE CONTAINED IN MYFILE
```

prints the list of functions in MYFILE.

```
WHAT RECORDS ARE ON MYFILE
```

prints the list of records.

(4) The implicit universe in which a set expression is interpreted depends on the type:

```
ANY VARIABLES @ GETD
```

is interpreted as the set of all variables which have been noticed by MasterScope (i.e., bound or used in any function which has been analyzed) that also have a definition.

```
ANY FUNCTIONS @ (NEQ (GETTOPVAL X) 'NOBIND)
```

is interpreted as the set of all functions which have been noticed (either analyzed or called by a function which has been analyzed) that also have a top-level value.

## Conjunctions of Sets

Sets may be joined by the conjunctions AND and OR or preceded by NOT to form new sets. AND is always interpreted as meaning intersection; OR as union; NOT as complement.

For example, the set CALLING X AND NOT CALLED BY Y specifies the set of all functions which call the function X but are not called by Y.

Note: MasterScope's interpretation of AND and OR follow Lisp conventions rather than the conventional English interpretation.

"Calling X and Y" would, in English, be interpreted as the intersection of (CALLING X) and (CALLING Y); but MasterScope interprets CALLING X AND Y as CALLING ('X AND 'Y), which is the null set.

Only sets may be joined with conjunctions. Joining modifiers, as in

```
USING X AS A RECORD FIELD OR PROPERTY NAME
```

is not allowed; in this case, you must type

```
USING X AS A RECORD FIELD OR USING X AS A
PROPERTY NAME
```

As described above, the type of set is used to disambiguate parsings. The algorithm used is to first try to match the type of the phrases being joined and then try to join with the longest preceding phrase.

In any case, you may group phrases with parentheses to specify the manner in which conjunctions should be parsed.

## SHOW PATHS

In trying to work with large programs, you can lose track of the hierarchy of functions. The MasterScope SHOW PATHS command aids you by providing a map showing the calling structure of a set of functions. SHOW PATHS prints out a tree structure showing which functions call which other functions.

Loading the Browser library module modifies the SHOW PATHS command so the command's output is displayed as an undirected graph.

The SHOW PATHS command takes the form: SHOW PATHS followed by some combination of the following path options:

FROM *SET* [MasterScope path option]

Display the function calls from the elements of *SET*.

TO *SET* [MasterScope path option]

Display the function calls leading to elements of *SET*. If TO is given before FROM (or no FROM is given), the tree is inverted and a message (inverted tree) is printed to warn you that if FN1 appears after FN2 it is because FN1 is called by FN2.

Note: When both FROM and TO are given, the first one indicates a set of functions which are to be displayed while the second restricts the paths that will be traced; i.e., the command SHOW PATHS FROM X TO Y will trace the elements of the set CALLED SOMEHOW BY X AND CALLING Y SOMEHOW.

If TO is not given, TO KNOWN OR NOT @ GETD is assumed; that is, only functions which have been analyzed or which are undefined will be included.

Note that MasterScope will analyze a function while printing out the tree if that function has not previously been seen and it currently has an EXPR definition. Thus, any function which can be analyzed will be displayed.

AVOIDING *SET* [MasterScope path option]

Do not display any function in *SET*. AMONG is recognized as a synonym for AVOIDING NOT.

For example, SHOW PATHS TO ERROR AVOIDING ON FILE2 will not display (or trace) any function on FILE2.

NOTRACE *SET* [MasterScope path option]

Do not trace from any element of *SET*. NOTRACE differs from AVOIDING in that a function which is marked NOTRACE will be printed, but the tree beyond it will not be expanded. The functions in an AVOIDING set will not be printed at all.

For example,

```
SHOW PATHS FROM ANY ON FILE1 NOTRACE ON FILE2
```

will display the tree of calls eminating from FILE1, but will not expand any function on FILE2.

SEPARATE *SET* [MasterScope path option]

Give each element of *SET* a separate tree.

Note: FROM and TO only insure that the designated functions will be displayed. SEPARATE can be used to guarantee that certain functions will begin new tree structures. SEPARATE functions are displayed in the same manner as overflow lines; i.e., when one of the functions indicated by SEPARATE is found, it is printed followed by a forward reference (a lower-case letter in braces) and the tree for that function is then expanded below.

LINELENGTH *N* [MasterScope path option]

Resets LINELENGTH to *N* before displaying the tree. The linelength is used to determine when a part of the tree should "overflow" and be expanded lower.

## Error Messages

When you give MasterScope a command, the command is first parsed, i.e. translated to an internal representation, and then the internal representation is interpreted.

If a command cannot be parsed, e.g. if you typed

```
SHOW WHERE CALLED BY X
```

MasterScope would reply

**Sorry, I can't parse that!**

and generate an error.

If the command is of the correct form but cannot be interpreted (e.g., the command EDIT WHERE ANY CONTAINS ANY) MasterScope will print the message

**Sorry, that isn't implemented!**

and generate an error.

If the command requires some functions having been analyzed (e.g., the command WHO CALLS X) and the data base is empty, MasterScope will print the message

**Sorry, no functions have been analyzed!**

and generate an error.

## Macro Expansion

As part of analysis, MasterScope will expand the macro definition of called functions if they are not otherwise defined (see *IRM*). MasterScope always expands Common Lisp DEFMACRO definitions (unless it finds a template for the macro).

MasterScope Interlisp macro expansion is controlled by a variable:

MSMACROPROPS [Variable]

Value is an ordered list of macro-property names that MasterScope will search to find a macro definition. Only the kinds of macros that appear on MSMACROPROPS will be expanded. All others will be treated as function calls and left unexpanded. Initially (MACRO).

Note: MSMACROPROPS initially contains only MACRO (not 10MACRO, DMACRO, etc.) on the assumption that the machine-dependent macro definitions are more likely "optimizers".

If you edit a macro, MasterScope will know to reanalyze the functions which call that macro.

Note: If your macro is of the "computed-macro" style, and it calls functions which you edit, MasterScope will not notice. You must be careful to tell masterscope to REANALYZE the appropriate functions (e.g., if you edit FOOEXPANDER which is used to expand FOO macros, you have to REANALYZE ANY CALLING FOO.

## Effecting MasterScope Analysis

MasterScope analyzes the EXPR definition of a function, and notes in its data base the relations that this function has with other functions and with variables. To perform this analysis, MasterScope uses templates which describe the behavior of functions.

For example, the information that SORT destructively modifies its first argument is contained in the template for SORT. MasterScope initially contains templates for most system functions that set variables, test their arguments, or perform destructive operations.

A template is a list structure containing any of the following atoms:

PPE [in MasterScope template]

If an expression appears in this location, there is most likely a parenthesis error.

MasterScope notes this as a call to the function ppe (lowercase). Therefore, SHOW WHERE ANY CALLS ppe will print out all possible parenthesis errors. When MasterScope finds a possible parenthesis error in the course of analyzing a function definition, rather than printing the usual ".", it prints out a "?" instead. MasterScope notes functions called with keywords they do not accept as calls to ppe.

NIL [in MasterScope template]

The expression occuring at this location is not evaluated.

SET [in MasterScope template]

A variable appearing at this place is set.

SMASH [in MasterScope template]

The value of this expression is smashed.

TEST [in MasterScope template]

Is used as a predicate (that is, the only use of the value of the expression is whether it is NIL or non-NIL).

PROP [in MasterScope template]

Is used as a property name. If the value of this expression is of the form (QUOTE *ATOM*), MasterScope will note that *ATOM* is USED AS A PROPERTY NAME.

For example, the template for GETPROP is (EVAL PROP . PPE).

KEYWORD key1... [in MasterScope template]

Must appear at the end of a template followed by the keywords the templated function accepts.

For example, the template for CL:MEMBER is (EVAL EVAL KEYWORDS :TEST :TEST-NOT :KEY).

FUNCTION [in MasterScope template]

The expression at this point is used as a functional argument.

For example, the template for MAPC is

`(SMASH FUNCTION FUNCTION . PPE).`

FUNCTIONAL [in MasterScope template]

The expression at this point is used as a functional argument. This is like FUNCTION, except that MasterScope distinguishes between functional arguments to functions which compile open from those that do not. For the latter (e.g. SORT and APPLY), FUNCTIONAL should be used rather than FUNCTION.

EVAL [in MasterScope template]

The expression at this location is evaluated (but not set, smashed, tested, used as a functional argument, etc.).

RETURN [in MasterScope template]

The value of the function (of which this is the template) is the value of this expression.

TESTRETURN [in MasterScope template]

A combination of TEST and RETURN: If the value of the function is non-NIL, then it is returned. For instance, a one-element COND clause is this way.

EFFECT [in MasterScope template]

The expression at this location is evaluated, but the value is not used. (That is, it is evaluated for its side effect only.)

FETCH [in MasterScope template]

An atom at this location is a field which is fetched.

REPLACE [in MasterScope template]

An atom at this location is a field which is replaced.

RECORD [in MasterScope template]

An atom at this location is used as a record name.

CREATE [in MasterScope template]

An atom at this location is a record which is created.

BIND [in MasterScope template]

An atom at this location is a variable which is bound.

CALL [in MasterScope template]

An atom at this location is a function which is called.

CLISP [in MasterScope template]

An atom at this location is used as a CLISP word.

! [in MasterScope template]

This atom, which can only occur as the first element of a template, allows you to specify a template for the CAR of the function form. If ! doesn't appear, the CAR of the form is treated as if it had a CALL specified for it. In other words, the templates (.. EVAL) and (! CALL .. EVAL) are equivalent.

If the next atom after a ! is NIL, this specifies that the function name should not be remembered.

For example, the template for AND is (! NIL .. TEST RETURN), which means that if you see an AND, don't remember it as being called. This keeps the MasterScope data base from being cluttered by too many uninteresting relations. MasterScope also throws away relations for COND, CAR, CDR, and a couple of others.

## Special Forms

In addition to the above atoms that occur in templates, there are some special forms which are lists keyed by their CAR.

.. *TEMPLATE* [in MasterScope template]

Any part of a template may be preceded by the atom .. (two periods) which specifies that the template should be repeated an indefinite number (*N*> = 0) of times to fill out the expression.

For example, the template for COND might be

```
(.. (TEST .. EFFECT RETURN))
```

while the template for SELECTQ is

```
(EVAL .. (NIL .. EFFECT RETURN) RETURN).
```

(Although MasterScope "throws away" the relations for COND, it makes sense to template COND because there may be important information within the arguments of COND.)

(BOTH *TEMPLATE1 TEMPLATE2*) [in MasterScope template]

Analyze the current expression twice, using the each of the templates in turn.

(IF *EXPRESSION TEMPLATE*$_1$ *TEMPLATE*$_2$) [in MasterScope template]

Evaluate *EXPRESSION* at analysis time (the variable EXPR will be bound to the expression which corresponds to the IF), and if the result is non-NIL, use *TEMPLATE1*, otherwise *TEMPLATE2*. If *EXPRESSION* is a literal atom, it is APPLYd to EXPR.

For example,

```
(IF LISTP (RECORD FETCH) FETCH)
```

specifies that if the current expression is a list, then the first element is a record name and the second element a field name, otherwise it is a field name.

(@ *EXPRFORM TEMPLATEFORM*) [in MasterScope template]

Evaluate *EXPRFORM* giving *EXPR*, evaluate *TEMPLATEFORM* giving *TEMPLATE*. Then analyze *EXPR* with *TEMPLATE*. @ lets you compute on the fly both a template and an expression to analyze with it. The forms can use the variable EXPR, which is bound to the current expression.

(MACRO . *MACRO*) [in MasterScope template]

*MACRO* is interpreted in the same way as macros (see *IRM*) and the resulting form is analyzed. If the template is the atom MACRO alone, MasterScope will use the MACRO property of the function itself. This is useful when analyzing code which contains calls to user-defined macros. If you change a macro property (e.g. by editing it) of an atom which has template of MACRO, MasterScope will mark any function which used that macro as needing to be reanalyzed.

Some examples of templates:

| Function: | Template: |
|---|---|
| DREVERSE | `(SMASH . PPE)` |
| AND | `(! NIL TEST .. RETURN)` |
| MAPCAR | `(EVAL FUNCTION FUNCTION)` |
| COND | `(! NIL .. (IF CDR (TEST .. EFFECT RETURN) (TESTRETURN . PPE)))` |

Templates may be changed and new templates defined using the following functions:

(GETTEMPLATE *FN*) [Function]

Returns the current template of *FN*.

(SETTEMPLATE *FN TEMPLATE*) [Function]

Changes the template for the function *FN* and returns the old value. If any functions in the data base are marked as calling *FN*, they will be marked as needing reanalysis.

## Updating the MasterScope Data Base

MasterScope is interfaced to the editor and file manager so that it notes whenever a function has been changed, either through editing or loading in a new definition. Whenever a command is given which requires knowing the information about a specific function, if that function has been noted as being changed, the function is automatically reanalyzed before the command is interpreted. If the command requires that all the information in the data base be consistent (e.g., you ask WHO CALLS X) then all functions which have been marked as changed are reanalyzed.

## MasterScope Entries

(MASTERSCOPE *COMMAND—)* [Function]

Top level entry to MasterScope. If *COMMAND* is NIL, will enter into an Executive in which you may enter commands. If *COMMAND* is not NIL, the command is interpreted and MASTERSCOPE will return the value that would be printed by the command.

Note that only the question commands return meaningful values.

(CALLS *FN USEDATABASE—*) [Function]

*FN* can be a function name, a definition, or a form.

Note: CALLS will also work on compiled code. CALLS returns a list of four elements:

Functions called by *FN*
Variables bound in *FN*
Variables used freely in *FN*
Variables used globally in *FN*

For the purpose of CALLS, variables used freely which are on GLOBALVARS or have a property GLOBALVAR value T are considered to be used globally. If *USEDATABASE* is NIL (or *FN* is not a symbol), CALLS will perform a one-time analysis of *FN*. Otherwise (i.e. if *USEDATABASE* is non-NIL and *FN* a function name), CALLS will use the information in MasterScope's data base (*FN* will be analyzed first if necessary).

(CALLSCCODE *FN*——) [Function]

The subfunction of CALLS which analyzes compiled code. CALLSCCODE returns a list of elements:

Functions called via "linked" function calls (not implemented in Interlisp-D)

Functions called regularly

Variables bound in *FN*

Variables used freely

Variables used globally

(FREEVARS *FN USEDATABASE*) [Function]

Equivalent to (CADDR (CALLS *FN USEDATABASE*)). Returns the list of variables used freely within *FN*.

(SETSYNONYM *PHRASE MEANING—*) [Function]

Defines a new synonym for MasterScope's parser. Both *OLDPHRASE* and *NEWPHRASE* are words or lists of words; anywhere *OLDPHRASE* is seen in a command, *NEWPHRASE* will be substituted.

For example,

```
(SETSYNONYM 'GLOBALS '(VARS IN GLOBALVARS OR
@(GETPROP X 'GLOBALVAR)))
```

would allow you to refer with the single word GLOBALS to the set of variables which are either in GLOBALVARS or have a GLOBALVAR property.

## Functions for Writing Routines

The following functions are provided for users who wish to write their own routines using MasterScope's data base:

(PARSERELATION *RELATION*) [Function]

*RELATION* is a relation phrase; e.g., (PARSERELATION '(USE FREELY)). PARSERELATION returns an internal representation for *RELATION*. For use in conjunction with GETRELATION.

(GETRELATION *ITEM RELATION INVERTED*) [Function]

*RELATION* is an internal representation as returned by PARSERELATION (if not, GETRELATION will first perform (PARSERELATION *RELATION*)).

*ITEM* is an atom. GETRELATION returns the list of all atoms which have the given relation to *ITEM*.

For example,

```
(GETRELATION 'X '(USE FREELY))
```

returns the list of variables that X uses freely.

If *INVERTED* is T, the inverse relation is used; e.g.

```
(GETRELATION 'X '(USE FREELY) T)
```

returns the list of functions which use X freely.

If *ITEM* is NIL, GETRELATION will return the list of atoms which have *RELATION* with *any* other item; i.e., it answers the question WHO *RELATIONS* ANY.

Note that GETRELATION does not check to see if *ITEM* has been analyzed, or that other functions that have been changed have been reanalyzed.

(TESTRELATION *ITEM RELATION ITEM2 INVERTED*) [Function]

Is equivalent to (MEMB *ITEM2* (GETRELATION *ITEM RELATION INVERTED*)); that is, it tests if *ITEM* and *ITEM2* are related via *RELATION*.

If *ITEM2* is NIL, the call is equivalent to

(NOT (NULL (GETRELATION *ITEM RELATION INVERTED*)))

i.e., TESTRELATION tests if *ITEM* has the given *RELATION* with any other item.

(MAPRELATION *RELATION MAPFN*) [Function]

Calls the function *MAPFN* on every pair of items related via *RELATION*. If (NARGS *MAPFN*) is 1, then *MAPFN* is called on every item which has the given *RELATION* to *any* other item.

(MSNEEDUNSAVE *FNS MSG MARKCHANGEFLG*) [Function]

Used to mark functions which depend on a changed record declaration (or macro, etc.), and which must be LOADed or UNSAVEd (see below). *FNS* is a list of functions to be marked, and *MSG* is a string describing the records, macros, etc. on which they depend. If *MARKCHANGEFLG* is non-NIL, each function in the list is marked as needing reanalysis.

(UPDATEFN *FN EVENIFVALID* —) [Function]

Equivalent to the command ANALYZE '*FN*; that is, UPDATEFN will analyze *FN* if *FN* has not been analyzed before or if it has been changed since the time it was analyzed. If *EVENIFVALID* is non-NIL, UPDATEFN will reanalyze *FN* even if MasterScope thinks it has a valid analysis in the data base.

(UPDATECHANGED) [Function]

Performs (UPDATEFN *FN*) on every function which has been marked as changed.

(MSMARKCHANGED *NAME TYPE REASON*) [Function]

Mark that *NAME* has been changed and needs to be reanalyzed. See MARKASCHANGED in the *IRM*.

(DUMPDATABASE *FNLST*) [Function]

Dumps the current MasterScope data base on the current output file in a LOADable form. If *FNLST* is not NIL, DUMPDATABASE will only dump the information for the list of functions in *FNLST*. The variable DATABASECOMS is initialized to

```
((E (DUMPDATABASE)))
```

Thus, you may merely perform (MAKEFILE 'DATABASE.*EXTENSION*) to save the current MasterScope data base. If a MasterScope data base already exists when a DATABASE file is loaded, the data base on the file will be merged with the one in memory.

Note: Functions whose definitions are different from their definition when the data base was made must be REANALYZEd if their new definitions are to be noticed.

Note: The DataBaseFns library module provides a more convenient way of saving data bases along with the source files to which they correspond.

## Noticing Changes that Require Recompiling

When a record declaration, iterative statement operator or macro is changed, and MasterScope has noticed a use of that declaration or macro (i.e. it is used by some function known about in the data base), MasterScope will alert you about those functions which might need to be recompiled (e.g. they do not currently have EXPR definitions). Extra functions may be noticed.

For example if FOO contains (fetch (REC X) --), and some declaration other than REC which contains X is changed, MasterScope will still think that FOO needs to be loaded/unsaved. The functions which need recompiling are added to the list MSNEEDUNSAVE and a message is printed out:

**The functions *FN1*, *FN2*,... use macros which have changed.**

Call UNSAVEFNS() to load and/or unsave them.

In this situation, the following function is useful:

(UNSAVEFNS —) [Function]

Uses LOADFNS or UNSAVEDEF to make sure that all functions in the list MSNEEDUNSAVE have EXPR definitions, and then sets MSNEEDUNSAVE to NIL.

Note: If RECOMPILEDEFAULT (see *IRM*) is set to CHANGES, UNSAVEFNS prints out

**"WARNING: you must set RECOMPILEDEFAULT to EXPRS in order to have these functions recompiled automatically."**

## Implementation Notes

MasterScope keeps a data base of the relations noticed when functions are analyzed. The relations are intersected to form primitive relationships such that there is little or no overlap of any of the primitives.

For example, the relation SET is stored as the union of SET LOCAL and SET FREE. The BIND relation is divided into BIND AS ARG, BIND AND NOT USE, and SET LOCAL, SMASH LOCAL, etc. Splitting the relations in this manner reduces the size of the data base considerably, to the point where it is reasonable to maintain a MasterScope data base for a large system of functions during a normal debugging session.

Each primitive relationship is stored in a pair of hash tables, one for the forward direction and one for the reverse.

For example, there are two hash tables, USE AS PROPERTY and USED AS PROPERTY. To retrieve the information from the data base, MasterScope performs unions of the hash values.

For example, to answer FOO BINDS WHO, MasterScope will look in all of the tables which make up the BIND relation. The internal representation returned by PARSERELATION is a list of dotted pairs of hash tables. To perform GETRELATION requires only mapping down that list, doing GETHASHs on the appropriate hash tables and UNIONing the result.

Hash tables are used for a variety of reasons: storage space is smaller; it is not necessary to maintain separate lists of which functions have been analyzed (a special table, DOESN'T DO ANYTHING is maintained for functions which neither call other functions nor bind or use any variables); and accessing is relatively fast. Within any of the tables, if the hash value is a list of one atom, then the atom itself, rather than the list, is stored as the hash value. This also reduces the size of the data base significantly.

## Example

### Sample Session

The following illustrates some of the MasterScope facilities.

```
50_. ANALYZE FUNCTIONS ON RECORD
...............................
NIL
51_. WHO CALLS RECFIELDLOOK
(RECFIELDLOOK ACCESSDEF ACCESSDEF2 EDITREC)
52_. EDIT WHERE ANY CALL RECFIELDLOOK
RECFIELDLOOK :
(RECFIELDLOOK (CDR Y) FIELD)
tty:
5*OK
ACCESSDEF :
(RECFIELDLOOK DECLST FIELD VAR1)
6*OK
(RECFIELDLOOK USERRECLST FIELD)
7*N VAR1
8*OK
ACCESSDEF2 :
(RECFIELDLOOK (RECORD.SUBDECS TRAN) FIELD)
tty:
(RECFIELDLOOK (RECORD.SUBDECS TRAN) FIELD)
9*N (CAR TAIL]
10*OK
EDITREC :
(RECFIELDLOOK USERRECLST (CAR EDITRECX))
11*OK
NIL
53_. WHO CALLS ERROR
..
(EDITREC)
54_. SHOW PATHS TO RECFIELDLOOK FROM ACCESSDEF
(inverted tree)
```

```
1. RECFIELDLOOK RECFIELDLOOK
2.                 ACCESSDEF
3.                 ACCESSDEF2 ACCESSDEF2
4.                                   ACCESSDEF
5.
RECORDCHAIN ACCESSDEF
NIL
55_. WHO CALLS WHO IN /FNS
RECORDSTATEMENT --  /RPLNODE
RECORDECL1 --       /NCONC, /RPLACD, /RPLNODE
RECREDECLARE1 --    /PUTHASH
UNCLISPTRAN --      /PUTHASH, /RPLNODE2
RECORDWORD --       /RPLACA
RECORD1 --          /RPLACA, /SETTOPVAL
EDITREC --          /SETTOPVAL
```

Event 50 You direct that the functions on file RECORD be analyzed. The leading period and space specify that this line is a MasterScope command. MasterScope prints a greeting and prompts with _. Within the top-level Executive of MasterScope, you may issue MasterScope commands, programmer's assistant commands, (e.g., REDO, FIX), or run programs. You can exit from the MasterScope Executive by typing OK. The function "." is defined as a Nlambda NoSpread function which interprets its argument as a MasterScope command, Executes the command and returns.

MasterScope prints a"." whenever it (re)analyzes a function, to let you know what it is happening. The feedback when MasterScope analyzes a function is controlled by the flag MSPRINTFLG: if MSPRINTFLG is the atom ".", MasterScope will print out a period. (If an error in the function is detected, "?" is printed instead.) If MSPRINTFLG is a number *N*, MasterScope will print the name of the function it is analyzing every *N*th function. If MSPRINTFLG is NIL, MasterScope won't print anything. Initial setting is ".".

Note that the function name is printed when MasterScope starts analyzing, and the comma is printed when it finishes.

Event 51 You ask which functions call RECFIELDLOOK. MasterScope responds with the list.

Statement 52 You ask to edit the expressions where the function RECFIELDLOOK is called. MasterScope calls EDITF on the functions it had analyzed that call RECFIELDLOOK, directing the editor to the appropriate expressions. You then edit some of those expressions. In this example, the teletype editor is used. If DEdit is enabled as the primary editor, it would be called to edit the appropriate functions.

Statement 53 Next you ask which functions call ERROR. Since some of the functions in the data base have been changed, MasterScope reanalyzes the changed definitions (and prints out .'s for each function it analyzes). MasterScope responds that EDITREC is the only analyzed function that calls ERROR.

Statement 54 You ask to see a map of the ways in which RECFIELDLOOK is called from ACCESSDEF. A tree structure of the calls is displayed.

Statement 55 You then ask to see which functions call which functions in the list /FNS. MasterScope responds with a structured printout of these relations.

## SHOW PATHS

The command SHOW PATHS FROM MSPARSE will print out the structure of MasterScope's parser:

```
1.MSPARSE  MSINIT MSMARKINVALID
2.         |      MSINITH MSINITH
3.         MSINTERPRET MSRECORDFILE
4.         |           MSPRINTWORDS
5.         |           PARSECOMMAND GETNEXTWORD CHECKADV
6.         |           |            PARSERELATION {a}
7.         |           |            PARSESET {b}
8.         |           |            PARSEOPTIONS {c}
9.         |           |            MERGECONJ GETNEXTWORD
{5}
10.        |           GETNEXTWORD {5}
11.        |           FIXUPTYPES SUBJTYPE
12.        |           |          OBJTYPE
13.        |           FIXUPCONJUNCTIONS MERGECONJ {9}
14.        |                             MATCHSCORE
15.        MSPRINTSENTENCE
------------------------------------------------------
overflow - a
16.PARSERELATION GETNEXTWORD {5}
17.              CHECKADV
------------------------------------------------------
overflow - b
19.PARSESET PARSESET
20.         GETNEXTWORD {5}
21.         PARSERELATION {6}
22.         SUBPARSE GETNEXTWORD {5}
------------------------------------------------------
overflow - c
23.PARSEOPTIONS GETNEXTWORD {5}
24.             PARSESET {19}
```

This example shows that the function MSPARSE calls MSINIT, MSINTERPRET, and MSPRINTSENTENCE. MSINTERPRET in turn calls MSRECORDFILE, MSPRINTWORDS, PARSECOMMAND, GETNEXTWORD, FIXUPTYPES, and FIXUPCONJUNCTIONS. The numbers in braces {} after a function name are backward references: they indicate that the tree for that function was expanded on a previous line. The lowercase letters in braces are forward references: they indicate that the tree for that function will be expanded below, since there is no more room on the line. The vertical bar is used to keep the output aligned.

# MATCH

Match provides a fairly general pattern match facility that allows you to specify certain tests that would otherwise be clumsy to write, by giving a pattern which the datum is supposed to match.

Essentially, you write "Does the (expression) X look like (the pattern) P?"

For example, (MATCH X WITH (& 'A -- 'B)) asks whether the second element of X is an A, and the last element a B.

## Requirements

DWIM must be enabled.

## Installation

Load MATCH.LCOM from the library.

## Programmer's Interface

(MATCH *OBJECT* WITH *PATTERN*) [CLISP operator]

Matches the *OBJECT* with the *PATTERN*.

The implementation of the matching is performed by computing (once) the equivalent Lisp expression which will perform the indicated operation, and substituting this for the pattern (rather than by invoking each time a general purpose capability such as that found in the AI languages FLIP or PLANNER).

For example, the translation of

```
(MATCH X WITH (& 'A -- 'B)) is:
(AND (EQ (CADR X) 'A)
     (EQ (CAR (LAST (CDDR X))) 'B))
```

Thus the pattern match facility is really a pattern match compiler, and the emphasis in its design and implementation has been more on the efficiency of object code than on generality and sophistication of its matching capabilities. The goal was to provide a facility that could and would be used even where efficiency was paramount, e.g., in inner loops. Wherever possible, already existing Lisp functions are used in the translation, e.g., the translation of ($ 'A $) uses MEMB, ($ ('A $) $) uses ASSOC, etc.

The syntax for pattern match expressions is (MATCH *FORM* WITH *PATTERN*), where *PATTERN* is a list as described below. If *FORM* appears more than once in the translation, and it is not either a variable or an expression that is easy to (re)compute, such as

(CAR Y), (CDDR Z), etc., a dummy variable will be generated and bound to the value of *FORM* so that *FORM* is not evaluated a multiple number of times.

For example, the translation of

`(MATCH (FOO X) WITH ($ 'A $))` is simply

`(MEMB 'A (FOO X)),`

while the translation of

`(MATCH (FOO X) WITH ('A 'B --))` is:

```
[PROG ($$2)
   (RETURN
      (AND (EQ (CAR (SETQ $$2 (FOO X))) 'A)
           (EQ (CADR $$2) 'B]
```

In the interests of efficiency, the pattern match compiler assumes that all lists end in NIL, i.e., there are no LISTP checks inserted in the translation to check tails.

For example, the translation of

`(MATCH X WITH ('A & --))` is

`(AND (EQ (CAR X) (QUOTE A)) (CDR X)),`

which will match with (A B) as well as (A . B).

Similarly, the pattern match compiler does not insert LISTP checks on elements, e.g.,

`(MATCH X WITH (('A --) --))` translates simply as

`(EQ (CAAR X) 'A),`

and

`(MATCH X WITH (($1 $1 --) --))` translates as

`(CDAR X).`

Note that you can explicitly insert LISTP checks yourself by using @, as described below, e.g.,

`(MATCH X WITH (($1 $1 --)@LISTP --))` translates as

`(CDR (LISTP (CAR X))).`

PATLISPCHECK [Variable]

The insertion of LISTP checks for *ELEMENTS* is controlled by the variable PATLISTPCHECK. When PATLISTPCHECK is T, LISTP checks are inserted, e.g.,

`(MATCH X WITH (('A --) --))` translates as:

`(EQ (CAR (LISTP (CAR (LISTP X)))) 'A).`

PATLISTPCHECK is initially NIL. Its value can be changed within a particular function by using a local CLISP declaration (see *IRM*).

PATVARDEFAULT [Variable]

Controls the treatment of !*ATOM* patterns (see below).

If PATVARDEFAULT is ' or QUOTE, !*ATOM* is treated the same as '*ATOM*.

If PATVARDEFAULT is = or EQUAL, same as = *ATOM*.

If PATVARDEFAULT is = = or EQ, same as = = *ATOM*.

If PATVARDEFAULT is _ or SETQ, same as *ATOM*_&.

PATVARDEFAULT is initially ' (quote).

PATVARDEFAULT can be changed within a particular function by using a local CLISP declaration (see *IRM*).

Note: Numbers and strings are always interpreted as though PATVARDEFAULT were =, regardless of its setting. EQ, MEMB, and ASSOC are used for comparisons involving small integers.

Note: Pattern match expressions are translated using the DWIM and CLISP facilities, using all CLISP declarations in effect (standard/fast/undoable; see *IRM*).

## Pattern Elements

A pattern consists of a list of pattern elements. Each pattern element is said to match either an element of a data structure or a segment.

For example, in the TTY editor's pattern matcher (see *IRM*), "--" matches any arbitrary segment of a list, while & or a subpattern match only one element of a list. Those patterns which may match a segment of a list are called segment patterns; those that match a single element are called element patterns.

### Element Patterns

There are several types of element patterns, best given by their syntax:

| | |
|---|---|
| $1 or & | Matches an arbitrary element of a list. |
| '*EXPRESSION* | Matches only an element which is equal to the given expression e.g., 'A, '(A B).<br><br>EQ, MEMB, and ASSOC are automatically used in the translation when the quoted expression is atomic, otherwise EQUAL, MEMBER, and SASSOC. |
| =*FORM* | Matches only an element which is EQUAL to the value of *FORM*; e.g., =X, =(REVERSE Y). |
| ==*FORM* | Same as =, but uses an EQ check instead of EQUAL. |
| *ATOM* | The treatment depends on setting of PATVARDEFAULT (see above). |
| (*PATTERN1* ... *PATTERNn*) | Matches a list which matches the given patterns; e.g., (& &), (-- 'A). |
| *ELEMENT-PATTERN*@*FN* | Matches an element if *ELEMENT-PATTERN* matches it, and *FN* (name of a function or a LAMBDA expression) applied to that element returns non-NIL.<br><br>For example, &@NUMBERP matches a number, and ('A --)@FOO matches a list whose first element is A and for which FOO applied to that list is non-NIL.<br><br>For simple tests, the function-object is applied before a match is attempted with the pattern, e.g., |

`((-- 'A --)@LISTP --)` translates as
`(AND (LISTP (CAR X)) (MEMB 'A (CAR X))),`
not the other way around. *FN* may also be a *FORM* in terms of the variable @, e.g., &@(EQ @ 3) is equivalent to = 3.

* Matches any arbitrary element. If the entire match succeeds, the element which matched the * will be returned as the value of the match.

Note: Normally, the pattern match compiler constructs an expression whose value is guaranteed to be non-NIL if the match succeeds and NIL if it fails. However, if a * appears in the pattern, the expression generated could also return NIL if the match succeeds and * was matched to NIL.

For example,
`(MATCH X WITH ('A * --))` translates as
`(AND (EQ (CAR X) 'A) (CADR X)),`
so if X is equal to (A NIL B) then (MATCH X WITH ('A * --)) returns NIL even though the match succeeded.

*~ELEMENT-PATTERN* Matches an element if the element is not (~) matched by *ELEMENT-PATTERN*, e.g., ~'A, ~ = X, ~(-- 'A --).

(*ANY* *ELEMENT-PATTERN ELEMENT-PATTERN* ...)

Matches if any of the contained patterns match.

## Segment Patterns

$ or -- Matches any segment of a list (including one of zero length).

The difference between $ and -- is in the type of search they generate.

For example,
`(MATCH X WITH ($ 'A 'B $))` translates as
`(EQ (CADR (MEMB 'A X)) 'B),` whereas
`(MATCH X WITH (-- 'A 'B $))` translates as:

```
[SOME X (FUNCTION (LAMBDA ($$2 $$1)
  (AND (EQ $$2 'A)
       (EQ (CADR $$1) 'B]
```

Thus, a paraphrase of ($ 'A 'B $) would be "Is B the element following the first A?", whereas a paraphrase of (-- 'A 'B $) would be "Is there any A immediately followed by a B?"

Note that the pattern employing $ will result in a more efficient search than that employing --. However, ($ 'A 'B $) will not match with (X Y Z A M O A B C), but (-- 'A 'B $) will.

Essentially, once a pattern following a $ matches, the $ never resumes searching, whereas -- produces a translation that will always continue searching until there is no possibility of success. However, if the pattern match compiler can deduce from the pattern that continuing a search after a particular failure cannot possibly succeed, then the translations for both -- and $ will be the same.

For example, both
`(MATCH X WITH ($ 'A $3 $))` and
`(MATCH X WITH (-- 'A $3 --))` translate as
`(CDDDR (MEMB (QUOTE A) X))`
because if there are not three elements following the first A, there certainly will not be three elements following subsequent A's, so there is no reason to continue searching, even for --.

Similarly, ($ 'A $ 'B $) and (-- 'A -- 'B --) are equivalent.

$2, $3, etc. Matches a segment of the given length.

Note that $1 is not a segment pattern.

!*ELEMENT-PATTERN* Matches any segment which *ELEMENT-PATTERN* would match as a list.

For example, if the value of FOO is (A B C), ! = FOO will match the segment ... A B C ... etc.

Note: Since ! appearing in front of the last pattern specifies a match with some tail of the given expression, it also makes sense in this case for a ! to appear in front of a pattern that can only match with an atom, e.g., ($2 !'A) means match if CDDR of the expression is the atom A.

Similarly,

`(MATCH X WITH ($ ! 'A))` translates to
`(EQ (CDR (LAST X)) 'A)`.

!*ATOM* The treatment depends on setting of PATVARDEFAULT.

If PATVARDEFAULT is ' or QUOTE, same as !'*ATOM* (see above discussion).

If PATVARDEFAULT is = or EQUAL, same as ! = *ATOM*.

If PATVARDEFAULT is = = or EQ, same as ! = = *ATOM*.

If PATVARDEFAULT is _ or SETQ, same as *ATOM*_$.

The atom "." is treated *exactly* like "!". In addition, if a pattern ends in an atom, the "." is first changed to "!", e.g., ($1 . A) and ($1 ! A) are equivalent, even though the atom "." does not explicitly appear in the pattern.

One exception where "." is not treated like "!" is when "." preceding an assignment does not have the special interpretation that "!" has preceding an assignment (see below).

For example,

`(MATCH X WITH ('A . FOO_'B))` translates as:
```
(AND (EQ (CAR X) 'A)
     (EQ (CDR X) 'B)
     (SETQ FOO (CDR X)))
```

but

```
(MATCH X WITH ('A ! FOO_'B)) translates as:
(AND (EQ (CAR X) 'A)
     (NULL (CDDR X))
     (EQ (CADR X) 'B)
     (SETQ FOO (CDR X)))
```

*SEGMENT-PATTERN@FUNCTION-OBJECT*

Matches a segment if the segment-pattern matches it, and the function object applied to the corresponding segment (as a list) returns non-NIL.

For example, ($@CDDR 'D $) matches (A B C D E) but not (A B D E), since CDDR of (A B) is NIL.

Note: An @ pattern applied to a segment will require computing the corresponding structure (with LDIFF) each time the predicate is applied (except when the segment in question is a tail of the list being matched).

## Assignments

Any pattern element may be preceded by "*VARIABLE_*", meaning that if the match succeeds (i.e., everything matches), *VARIABLE* is to be set to the thing that matches that pattern element.

For example, if X is (A B C D E), (MATCH X WITH ($2 Y_$3)) will set Y to (C D E).

Note that assignments are not performed until the entire match has succeeded, so assignments cannot be used to specify a search for an element found earlier in the match, e.g., (MATCH X WITH (Y_$1 = Y --)) will not match with (A A B C ...), unless, of course, the value of Y was A before the match started. This type of match is achieved by using place-markers, described below.

If the variable is preceded by a !, the assignment is to the tail of the list as of that point in the pattern, i.e., that portion of the list matched by the remainder of the pattern.

For example, if X is (A B C D E), (MATCH X WITH ($ !Y_'C 'D $)) sets Y to (C D E), i.e., CDDR of X. In other words, when ! precedes an assignment, it acts as a modifier to the _, and has no effect whatsoever on the pattern itself, e.g., (MATCH X WITH ('A 'B)) and (MATCH X WITH ('A !FOO_'B)) match identically, and in the latter case, FOO will be set to CDR of X.

Note: *_*PATTERN-ELEMENT* and !*_*PATTERN-ELEMENT* are acceptable, e.g.,

```
(MATCH X WITH ($ 'A *_('B --) --)) translates as:
[PROG ($$2) (RETURN
 (AND (EQ (CAADR (SETQ $$2 (MEMB 'A X)))  'B)
      (CADR $$2]
```

## Place Markers

Variables of the form *#N*, where *N* is a number, are called place markers, and are interpreted specially by the pattern match compiler. Place markers are used in a pattern to mark or refer to a particular pattern element. Functionally, they are used like ordinary variables, i.e., they can be assigned values, or used freely in forms appearing in the pattern.

For example,

```
(MATCH X WITH (#1_$1 =(ADD1 #1)))
```

will match the list (2 3).

However, they are not really variables in the sense that they are not bound, nor can a function called from within the pattern expect to be able to obtain their values. For convenience, regardless of the setting of PATVARDEFAULT, the first appearance of a defaulted place-marker is interpreted as though PATVARDEFAULT were __.

Thus the above pattern could have been written as

```
(MATCH X WITH ( 1 =(ADD1  1))).
```

Subsequent appearances of a place-marker are interpreted as though PATVARDEFAULT were =.

For example,

`(MATCH X WITH (#1 #1 --))` is equivalent to
`(MATCH X WITH (#1_$1 =#1 --))`, and translates as
`(AND (CDR X) (EQUAL (CAR X) (CADR X))`.

Note that (EQUAL (CAR X) (CADR X)) would incorrectly match with (NIL).

## Replacements

The construct *PATTERN-ELEMENT_FORM* specifies that if the match succeeds, the part of the data that matched is to be replaced with the value of *FORM*.

For example, if X = (A B C D E), (MATCH X WITH ($ 'C $1_Y $1)) will replace the third element of X with the value of Y. As with assignments, replacements are not performed until after it is determined that the entire match will be successful.

Replacements involving segments splice the corresponding structure into the list being matched, e.g., if X is (A B C D E F) and FOO is (1 2 3), after the pattern ('A $_FOO 'D $) is matched with X, X will be (A 1 2 3 D E F), and FOO will be EQ to CDR of X, i.e., (1 2 3 D E F).

Note that ($ FOO_FIE $) is ambiguous, since it is not clear whether FOO or FIE is the pattern element, i.e., whether _ specifies assignment or replacement.

For example, if PATVARDEFAULT is =, this pattern can be interpreted as ($ FOO_ = FIE $), meaning search for the value of

FIE, and if found set FOO to it, or ($ =FOO_FIE $) meaning search for the value of FOO, and if found, store the value of FIE into the corresponding position. In such cases, you should disambiguate by not using the PATVARDEFAULT option, i.e., by specifying ' or =.

Note: Replacements are normally done with RPLACA or RPLACD. You can specify that /RPLACA and /RPLACD should be used, or FRPLACA and FRPLACD, by means of CLISP declarations (see *IRM*).

## Reconstruction

You can specify a value for a pattern match operation other than what is returned by the match by writing (MATCH *FORM1* WITH *PATTERN* = > *FORM2*).

For example,

```
(MATCH X WITH (FOO_$ 'A --) => (REVERSE FOO))
```

translates as:

```
[PROG ($$2)
   (RETURN
      (COND ((SETQ $$2 (MEMB 'A X))
             (SETQ FOO (LDIFF X $2))
             (REVERSE FOO]
```

Place markers in the pattern can be referred to from within *FORM*, e.g., the above could also have been written as

```
(MATCH X WITH (!#1 'A --) => (REVERSE #1)).
```

If -> is used in place of = >, the expression being matched is also physically changed to the value of *FORM*.

For example,

```
(MATCH X WITH (#1 'A !#2) -> (CONS #1 #2))
```

would remove the second element from X, if it were equal to A.

In general, (MATCH *FORM1* WITH *PATTERN* -> *FORM2*) is translated so as to compute *FORM2* if the match is successful, and then smash its value into the first node of *FORM1*. However, whenever possible, the translation does not actually require *FORM2* to be computed in its entirety, but instead the pattern match compiler uses *FORM2* as an indication of what should be done to *FORM1*.

For example,

```
(MATCH X WITH (#1 'A !#2) -> (CONS #1 #2))
```

translates as

```
(AND (EQ (CADR X) 'A) (RPLACD X (CDDR X))).
```

## Limitations

The pattern match facility does not contain some of the more esoteric features of other pattern match languages, such as repeated patterns, disjunctive and conjunctive patterns, recursion, etc. However, you can be confident that what facilities it does provide will result in Lisp expressions comparable to those you would generate by hand.

## Examples

```
(MATCH X WITH (-- 'A --))
```

-- matches any arbitrary segment. 'A matches only an A, and the second -- again matches an arbitrary segment; thus this translates to (MEMB 'A X).

```
(MATCH X WITH (-- 'A))
```

Again, -- matches an arbitrary segment; however, since there is no -- after the 'A, A must be the last element of X. Thus this translates to: `(EQ (CAR (LAST X)) 'A)`.

```
(MATCH X WITH ('A 'B -- 'C $3 --))
```

CAR of X must be A, and CADR must be B, and there must be at least three elements after the first C, so the translation is:

```
(AND (EQ (CAR X) 'A)
     (EQ (CADR X) 'B)
     (CDDDR (MEMB 'C (CDDR X))))
```

```
(MATCH X WITH (('A 'B) 'C Y_$1 $))
```

Since ('A 'B) does not end in $ or --, (CDDAR X) must be NIL. The translation is:

```
(COND
   ((AND (EQ (CAAR X) 'A)
         (EQ (CADAR X) 'B)
         (NULL (CDDAR X))
         (EQ (CADR X) 'C)
         (CDDR X))
    (SETQ Y (CADDR X)) T))
```

```
(MATCH X WITH (#1 'A $ 'B 'C #1 $))
```

#1 is implicitly assigned to the first element in the list. The $ searches for the first B following A. This B must be followed by a C, and the C by an expression equal to the first element. The translation is:

```
[PROG ($$2)
   (RETURN
      (AND (EQ (CADR X) 'A)
           (EQ [CADR (SETQ $$2 (MEMB 'B (CDDR X] 'C)
           (CDDR $$2)
           (EQUAL (CADDR $$2) (CAR X]
```

```
(MATCH X WITH (#1 'A -- 'B 'C #1 $))
```

Similar to the pattern above, except that -- specifies a search for *any* B followed by a C followed by the first element, so the translation is:

```
[AND (EQ (CADR X) 'A)
     (SOME (CDDR X)
           (FUNCTION (LAMBDA ($$2 $$1)
              (AND (EQ $$2 'B)
                   (EQ (CADR $$1) 'C)
                   (CDDR $$1)
                   (EQUAL (CADDR $$1) (CAR X]
```

# MATMULT

Two dimensional graphical transformations, such as rotations, scalings, and translations are conveniently represented as homogeneous 3-by-3 matrices, which operate on homogeneous 3-vectors. Similarly, three dimensional graphical transformations are conveniently represented as homogeneous 4-by-4 matrices, which operate on homogeneous 4-vectors. MatMult provides utilities for creating and manipulating such matrices and vectors, and takes advantage of microcode support for high-speed 3-by-3 and 4-by-4 matrix multiplication.

All matrices and vectors in MatMult are represented as Common Lisp arrays of element type single-float, so the Common Lisp array functions are sufficient to create and access individual elements of these specialized arrays. However, MatMult provides convenient wrapper functions for most common operations on these arrays.

All the following functions that return arrays accept optional array arguments. If given a result argument, these functions alter the contents of that argument rather then allocating new storage. It is an error for the optional array argument to be not of element type single-float, or to have incorrect dimensions.

## Requirements

MatMult should be run on an 1109 with a Weitek floating point chip set, but is also quite efficient on an 1186.

## Installation

Load MATMULT.LCOM from the library.

## Matrix Creation Functions

(MAKE-HOMOGENEOUS-3-VECTOR *X Y*) [Function]

Returns a 3-vector of element type single-float. If *X* or *Y* is provided, then the corresponding element of the vector is set appropriately, otherwise it defaults to 0.0. The third element of the vector is always initialized to 1.0.

Note: Throughout this text, "set" is used to emphasize that the value of the result element is altered and that no new storage is allocated to it.

(MAKE-HOMOGENEOUS-3-BY-3 *&KEY A00 A01 A10 A20 A21*) [Function]

Returns a 3-by-3 matrix of element type single-float. If a keyword argument is provided, the corresponding element of the matrix is set appropriately, otherwise entries default to 0.0. The (2 ,2) is always initialized to 1.0.

(MAKE-HOMOGENEOUS-N-BY-3 *N &KEY INITIAL-ELEMENT*) [Function]

Returns an N-by-3 matrix of element type single-float. If the keyword argument is provided, all the elements in the first two columns are set appropriately, otherwise they default to 0.0. The third column is always initialized to 1.0.

(MAKE-HOMOGENEOUS-4-VECTOR *X Y Z*) [Function]

Returns a 4-vector of element type single-float. If *X*, *Y* or *Z* is provided then the corresponding element of the vector is set appropriately, otherwise it defaults to 0.0. The forth element of the vector is always initialized to 1.0.

(MAKE-HOMOGENEOUS-4-BY-4 *&KEY A00 A01 A02 A03 A10 A11 A12 A13 A20 A21 A22 A23 A30 A31 A32* [Function]

Returns a 4-by-4 matrix of element type single-float. If a keyword arguments is provided, the corresponding element of the matrix is set appropriately, otherwise entries default to 0.0. The (3 ,3) is always initialized to 1.0.

(MAKE-HOMOGENEOUS-N-BY-4 *N &KEY INITIAL-ELEMENT*) [Function]

Returns an N-by-4 matrix of element type single-float. If the keyword argument is provided, all the elements in the first three columns are set appropriately, otherwise they default to 0.0. The forth column is always initialized to 1.0.

(IDENTITY-3-BY-3 *RESULT*) [Function]

Returns a 3-by-3 identity matrix.

If *RESULT* is supplied, it is side effected and returned.

(That is, the storage associated with the optional result argument is reused for the result, rather than allocating new storage for the result.)

(IDENTITY-4-BY-4 *RESULT*) [Function]

Returns a 4-by-4 identity matrix. If *RESULT* is supplied, it is side effected and returned.

(ROTATE-3-BY-3 *RADIANS RESULT*) [Function]

Returns a 3-by-3 rotation matrix specified by a counter-clockwise rotation of *RADIANS* radians. If *RESULT* is supplied, it is set and returned.

(ROTATE-4-BY-4-ABOUT-X *RADIANS RESULT*) [Function]

Returns a 4-by-4 rotation matrix specified by a positive right-handed rotation of *RADIANS* radians about the X axis. If *RESULT* is supplied, it is set and returned.

(ROTATE-4-BY-4-ABOUT-Y *RADIANS RESULT*) [Function]

Returns a 4-by-4 rotation matrix specified by a positive right-handed rotation of *RADIANS* radians about the Y axis. If *RESULT* is supplied, it is set and returned.

(ROTATE-4-BY-4-ABOUT-Z *RADIANS RESULT*) [Function]

Returns a 4-by-4 rotation matrix specified by a positive right-handed rotation of *RADIANS* radians about the Z axis. If *RESULT* is supplied, it is set and returned.

(SCALE-3-BY-3 *SX SY RESULT*) [Function]

Returns a 3-by-3 homogeneous scaling transformation that scales by a factor of *SX* along the X-axis and *SY* along the Y-axis. If *RESULT* is supplied, it is set and returned.

(SCALE-4-BY-4 *SX SY SZ RESULT*) [Function]

Returns a 4-by-4 homogeneous scaling transformation that scales by a factor of *SX* along the X-axis, *SY* along the Y-axis, and *SZ* along the Z axis. If *RESULT* is supplied, it is set and returned.

(TRANSLATE-3-BY-3 *TX TY RESULT*) [Function]

Returns a 3-by-3 homogeneous translation that translates by *TX* along the X-axis and *TY* along the Y-axis. If *RESULT* is supplied, it is set and returned.

(TRANSLATE-4-BY-4 *TX TY TZ RESULT*) [Function]

Returns a 4-by-4 homogeneous translation that translates by *TX* along the X-axis, *TY* along the Y-axis and *TZ* along the Z axis. If *RESULT* is supplied, it is set and returned.

(PERSPECTIVE-4-BY-4 *PX PY PZ RESULT*) [Function]

Returns a 4-by-4 homogeneous perspective transformation defined by *PX, PY*, and *PZ*. If *RESULT* is supplied, it is set and returned.

## Matrix Multiplication Functions

If run on workstations equipped with the extended processor option, these functions make good use of the hardware floating-point unit. The three digits at the end of each function's name describe the dimensions of their arguments.

Note: The results of the following matrix multiplication functions are not guaranteed to be correct unless the matrix arguments are all different (Not EQ).

(MATMULT-133 *VECTOR MATRIX RESULT*) [Function]

Returns the inner product of a 3-vector, *VECTOR*, and a 3-by-3 matrix, *MATRIX*. If *RESULT* is supplied, it is set and returned.

(MATMULT-331 *MATRIX VECTOR RESULT*) [Function]

Returns the inner product of a 3-by-3 matrix, *MATRIX*, and a 3-vector, *VECTOR*. If *RESULT* is supplied, it is set and returned.

(MATMULT-333 *MATRIX-1 MATRIX-2 RESULT*) [Function]

Returns the inner product of a 3-by-3 matrix, *MATRIX-1*, and another 3-by-3 matrix, *MATRIX-2*. If *RESULT* is supplied, it is set and returned.

(MATMULT-N33 *MATRIX-1 MATRIX-2 RESULT*) [Function]

Returns the inner product of an N-by-3 matrix, *MATRIX-1*, and a 3-by-3 matrix, *MATRIX-2*. If *RESULT* is supplied, it is set and returned.

(MATMULT-144 *VECTOR MATRIX RESULT*) [Function]

Returns the inner product of a 4-vector, *VECTOR*, and a 4-by-4 matrix, *MATRIX*. If *RESULT* is supplied, it is set and returned.

(MATMULT-441 *MATRIX VECTOR RESULT*) [Function]

Returns the inner product of a 4-by-4 matrix, *MATRIX*, and a 4-vector, *VECTOR*. If *RESULT* is supplied, it is set and returned.

(MATMULT-444 *MATRIX-1 MATRIX-2 RESULT*) [Function]

Returns the inner product of a 4-by-4 matrix, *MATRIX-1*, and another 4-by-4 matrix, *MATRIX-2*. If *RESULT* is supplied, it is set and returned.

(MATMULT-N44 *MATRIX-1 MATRIX-2 RESULT*) [Function]

Returns the inner product of an N-by-4 matrix, *MATRIX-1*, and a 4-by-4 matrix, MATRIX-2. If *RESULT* is supplied, it is set and returned.

## Miscellaneous Functions

(PROJECT-AND-FIX-3-VECTOR *3-VECTOR 2-VECTOR*) [Function]

The homogeneous *3-VECTOR* is projected onto the X-Y plane, coerced to integer coordinates (rounding by truncation) and returned. If *2-VECTOR* is supplied, it is set and returned.

(PROJECT-AND-FIX-N-BY-3 *N-3-MATRIX N-2-MATRIX*) [Function]

The homogeneous N-by-3 matrix, *N-3-MATRIX*, is projected onto the X-Y plane row-by-row, coerced to integer coordinates (rounding by truncation) and returned. If *N-2-MATRIX* is supplied, it is set and returned.

(PROJECT-AND-FIX-4-VECTOR *4-VECTOR 2-VECTOR*) [Function]

The homogeneous 4-vector, *4-VECTOR*, is projected onto the X-Y plane, coerced to integer coordinates (rounding by truncation) and returned. If *2-VECTOR* is supplied, it is set and returned.

(PROJECT-AND-FIX-N-BY-4 *N-4-MATRIX N-2-MATRIX*) [Function]

The homogeneous N-by-4 MATRIX, *N-3-MATRIX*, is projected onto the X-Y plane row-by-row, coerced to integer coordinates (rounding by truncation) and returned. If *N-2-MATRIX* is supplied, it is set and returned.

(DEGREES-TO-RADIANS *DEGREES*) [Function]

Returns *DEGREES* converted to radians.

## Limitations

MatMult is not intended as a general matrix manipulation package; it is specialized for the 3-by-3 and 4-by-4 cases.

Use CmlFloatArray for more general floating point array facilities.

## Example

```
(* ; "Try (spiral)")


(CL:DEFUN SPIRAL (&OPTIONAL (WINDOW (CREATEW))
                       &AUX
                       (WIDTH (WINDOWPROP WINDOW 'WIDTH))
                       (HALF-WIDTH (QUOTIENT WIDTH 2))
                       (HEIGHT (WINDOWPROP WINDOW 'HEIGHT))
                       (HALF-HEIGHT (QUOTIENT HEIGHT 2))
                       (SCALE-FACTOR (CL:EXP (QUOTIENT
                            (CL:LOG (QUOTIENT (MIN WIDTH HEIGHT) 2.0))
1440.0))))
   (LET ((LINE-1 (MAKE-HOMOGENEOUS-3-VECTOR 1.0 0.0))
         (LINE-2 (MAKE-HOMOGENEOUS-3-VECTOR))
         (TEMP (MAKE-HOMOGENEOUS-3-VECTOR))
         (POINTS (CL:MAKE-ARRAY 2))
         (TRANSFORM (MATMULT-333 (ROTATE-3-BY-3 (DEGREES-TO-RADIANS 2.5))
                          (SCALE-3-BY-3 SCALE-FACTOR SCALE-FACTOR)))
         (TRANSLATION (TRANSLATE-3-BY-3 HALF-WIDTH HALF-HEIGHT)))
        (CL:DO ((L-1 LINE-1)
                (L-2 LINE-2)
                (I 0 (CL:1+ I)))
               ((EQ I 1728))
               (MATMULT-133 L-1 TRANSFORM L-2)
               (MATMULT-133 L-2 TRANSLATION TEMP)
               (PROJECT-AND-FIX-3-VECTOR TEMP POINTS)
               (DRAWLINE HALF-WIDTH HALF-HEIGHT (CL:AREF POINTS 0)
                      (CL:AREF POINTS 1)
                      1
                      'REPLACE WINDOW)
               (CL:ROTATEF L-1 L-2))))
```

[This page intentionally left blank]

# MINISERVE

MiniServe contains servers for three simple protocols: Time Service (both PUP and XNS versions) and PUP ID Service. The servers are intended to run in the background on an 1108 or 1186 on networks that lack other sources of these services.

## Requirements

The time must be correctly set on the machine running MiniServe (see "NS Time Service" below).

## Installation

Load MINISERVE.LCOM from the library.

Either set the variable NS.TO.PUP.ALIST correctly, or make sure that the variable NS.TO.PUP.FILE is the name of a file containing a single form which will be used to set NS.TO.PUP.ALIST (see "PUP ID Service" below).

Evaluate (STARTMINISERVER).

## Functions

(STARTMINISERVE) [Function]

This function has no arguments; it adds three background processes to the environment, one for each of the protocols that miniserve handles. These processes and protocols are:

| | |
|---|---|
| \NSTIMESERVER | Provides the XNS Time Service |
| \PUPTIMESERVER | Provides the PUP Time Service |
| \PUP.ID.SERVER | Provides the PUP ID Service |

### XNS Time Service

XNS Time Service answers requests for the time using the XNS Time Protocol.

You must already have set the correct date and time on your workstation, either via one of the installation utilities or by evaluating

```
(SETTIME "dd-MMM-yy hh:mm:ss").
```

If you are not in the Pacific time zone, you should also make sure the following variables are set correctly:

\BEGINDST [Variable]

The ordinal day of the year (1 = January 1, 366 = December 31) on or before which daylight saving time starts in your area. Set it to 367 if your area does not observe daylight saving time.

\ENDDST [Variable]

The ordinal day of the year on or before which daylight saving time ends.

\TIMEZONECOMP [Variable]

The number of hours west of Greenwich; e.g., Eastern standard time = 5.

## PUP Time Service

PUP Time Service is like NS Time Service, but using a PUP protocol. This service is not required by any Xerox workstation as long as XNS Time Service is available, but may be of use to other workstations.

You can disable it by evaluating

```
(MOVD 'NILL '\PUPTIMESERVER).
```

## PUP ID Service

PUP ID Service supplies workstations with PUP host numbers, given their 48-bit XNS host numbers, so that they may communicate via PUP protocols.

NS.TO.PUP.FILE [Variable]

The name of a file containing a single form which will be used to set NS.TO.PUP.ALIST. Either this variable or NS.TO.PUP.ALIST must be set for the PUP ID Service to work.

NS.TO.PUP.ALIST [Variable]

A list which maps a workstation's XNS host number to a pup host number. Elements of this list are dotted pairs of the form:

```
((NSHOSTNUMBER A B C) . PUPNUMBER)
```

where *A, B, C* are the three 16-bit components of the workstation's 48-bit XNS host number (the value of the variable \MY.NSHOSTNUMBER), and PUPNUMBER is the corresponding PUP host number to be assigned to the workstation. PUP host numbers are integers in the range [1,254], and must be unique among hosts on a single net.

To set up this list correctly you can do the following on each workstation which will use the service (including the workstation running MiniServe):

1. Decide on a unique PUP host number for this workstation. It must be an integer inthe range [1,254]. For example we'll choose PUP Host number 2.

2. Get the workstation's NS host number and add it to the PUP host number. Evaluate the following form:

   (CONS \MY.NSHOSTNUMBER *YOURPUPNUMBER*)

   Using our chosen PUP host number of "2" and an example value for \MY.NSHOSTNUMBER the result might be:

   ```
   ((NSHOSTNUMBER 0 43520 14312) . 2)
   ```

3. Back on the workstation which is about to run MINISERVE, insert the dotted pair into NS.TO.PUP.ALIST.

## Restarting MiniServe

If you need to restart MiniServe:

Use the PSW window to kill the three processes that were started by STARTMINISERVE.

Evaluate (STARTMINISERVE).

[This page intentionally left blank]

# NSMAINTAIN

NSMaintain allows you to view and modify objects in the Clearinghouse data base from inside Lisp. Similar operations are available when chatting to a Clearinghouse service.

## Requirements

Xerox NS network environment with Clearinghouse server(s).

DES.LCOM.

## Installation

Load NSMAINTAIN.LCOM from the library. This file automatically loads DES.LCOM. DES is currently only used by the Change Password command, so its loading can be omitted if you do not need that command.

## Clearinghouse Concepts

The Clearinghouse maintains a distributed data base of *objects*, each of which has a set of *properties*. The objects are such things as users, groups, and network servers; the properties are such attributes as a server address or a user's mailbox location.

Clearinghouse objects are partitioned into a three-level hierarchy: each object is contained in a *domain*, which in turn is part of an *organization*. A fully qualified object name is a three-part name in the form *object:domain:organization*. Similarly, a domain name is a two-part name of the form *domain:organization*. Lisp maintains a notion of the *default domain*, which is typically the domain in which you and the servers in your immediate area are registered. When typing object names, you may omit the *organization* field or both *domain* and *organization* fields if they are the same as your default domain. Similarly, when typing a domain name, you may omit the *organization* field if it is the same as the default.

When printing the names of objects, the system usually elides the domain and/or organization, following the same rules. For example, for the object named "John Jones:Sales:ACME", the system would print "John Jones:" if the default domain were "Sales:ACME", or "John Jones:Sales" if the default domain were "Admin:ACME". NSMaintain, however, prints fully qualified names in certain places that do not need the compactness of the elided names, so as to reduce potential confusion. For the same reason, whenever NSMaintain prompts for an object name and you omit one or two of the fields, NSMaintain automatically echoes the defaults for you. You can change the defaults with the "Change Default Domain" command.

Any object in the Clearinghouse can have one or more *aliases*, which are Clearinghouse names that point directly to the object. An alias can be thought of as a "nickname", and can be used interchangeably with the "real name" for virtually all operations. For example, it is common practice to register users with their full names and provide at least one alias consisting of their last names.

Some objects in the Clearinghouse are groups, rather than individuals. Groups are described further in the section on group commands.

For more information on the Clearinghouse service, consult the *Interlisp-D Reference Manual* or the Clearinghouse documentation, which is part of the Network Systems documentation kit.

## User Interface

NSMaintain runs in an Exec window. To start it up, evaluate:

(NSMAINTAIN) [Function]

Starts an NSMaintain session. It prompts with "CH:" in the current window and awaits commands from you. Command names complete automatically following one- or two-letter inputs. Type Q, for the Quit command, when you wish to finish.

Most of the commands take as input from you one or more Clearinghouse object names. For many commands, NSMaintain offers you the same name as you last used in a similar context. For example, if you use the Describe command to learn about an object that is a group and then use the List Members command, NSMaintain offers you the group name just described. To accept the name, just press the carriage return; otherwise, start typing the desired name; your type-in replaces the offered name. The alphabetic case of names is not significant; you may type in either upper or lowercase. However, the commands that create objects will preserve the exact case of the name as you first type it.

Typing a null name to most commands aborts the command. If a sample name is offered, you have to backspace over it, or use Control-Q to erase the whole name. You can also usually use Control-E to abort a command.

The description of the commands below is partitioned into two parts: general *user* commands and *administrator* commands. The user commands can be used by anyone, and mostly are concerned with viewing the data base. The administrator commands allow system administrators to modify the data base; these commands cannot be used by ordinary users. However, there are two administrator commands, Add Self and Remove

Self, that can be used by anybody to join or leave groups with open access.

## User Commands

Anyone can use these commands to obtain information, change passwords, and change NSMaintain's defaults.

### Obtaining Information

These commands let you examine the data base.

Most of the List commands enumerate items in the data base matching a particular *pattern*. A pattern is a Clearinghouse name optionally containing asterisks as wild cards, which match zero or more characters. Wild cards are permitted only in the first component of the name; the remaining parts must be a valid domain and organization. In a two-part name, wild cards are permitted in the domain name but not the organization. Thus, for example, "*John*:Sales:ACME" is a valid pattern matching objects whose name component contains the substring "John"; "Joe:*:ACME" is not a valid pattern.

Following a List command (except List Domains), you can use the Show Details command to get more information about any of the names listed.

| | |
|---|---|
| Describe | Gives a description of any object registered in the Clearinghouse, what its registered name is (in case you typed an alias), and all interesting properties of the object. If the object is a group, its Owner and Friends are also listed; to see its members, use the List Members command. |
| List Domains | Lists all domains matching a specified domain pattern; for example, "*:Xerox" to list all domains in the Xerox organization. |
| List Clearinghouses | Lists all Clearinghouse servers that serve a specified domain. |
| List Administrators | Lists the administrators for a domain. |
| List Aliases | Lists all aliases matching a given pattern. Note that none of the other List commands (except for List Objects with property any) will match your pattern against an alias, so you may want to use the List Aliases command if you don't find the object you were looking for otherwise. |
| List Groups | Lists all groups matching a given pattern. |
| List True Groups | Same as List Groups, but filters out all names that also have a "user" property. These "groups" are typically used for mail forwarding. This command requires considerably more computation than List Groups. |
| List Servers | Lists objects matching a given pattern and registered as a server. You are prompted for the type of server; for example, Mail, File, or Print. Type ? to see the choices. |
| List Users | Lists the names of all users matching a given pattern. |
| List Objects | Lists all registered objects of of an arbitrary Clearinghouse type that match a given pattern. You are prompted for the type(a Clearinghouse property name) and pattern. To list all objects, |

that is, those with *any* property, press the carriage return to the property prompt, or supply the property "*".

| | |
|---|---|
| List Members | Lists the members of a specified group. |
| Show Details | Prompts you for a name, performing automatic spelling completion from the names printed in the most recent List command such as List Users, and then performs the Describe command on the name. Press the carriage return in response to the name prompt to return to the main CH: prompt. If there was only one name in the list, this command does a Describe on it without further prompting. |
| Type Entry | Synonym for Describe. |
| Type Members | Synonym for List Members. |

### Miscellaneous

| | |
|---|---|
| Change Default Domain | Changes the defaults used on type-in inside NSMaintain for domain and organization. NSMaintain asks if you also want to change the defaults globally; if you say yes, the variables CH.DEFAULT.DOMAIN and CH.DEFAULT.ORGANIZATION are changed, so that the new defaults have effect outside of NSMaintain as well. The defaults are never used when typing aliases in the Add Alias and Add User commands; these commands default the domain to be the same as the domain of the main object. Note: Unless you change the defaults globally, this command does not affect type-out and the Change Login command, which are still performed with respect to the global defaults. This may change in a future implementation. |
| Change Login | Prompts you for a new name and password, which becomes the default NS login on your machine and for NSMaintain. You can also use this to fix your password if you were incorrectly logged in before you started NSMaintain. |
| Change Password | Allows you to change your password. Prompts for a user name, offering your logged-in name as default. A domain administrator can also change other users' passwords. After you type the new password, you are asked to retype the password, to ensure that you typed what you thought you had. Neither password is echoed. |
| Quit | Exit NSMaintain. |

## Administrator Commands

These commands modify the Clearinghouse data base. In general, they require that you have the appropriate administrator access.

### Creating and Deleting Objects

To create or delete objects in a domain, you must be an administrator for the domain.

| | |
|---|---|
| Add Alias | Assigns an alias for a specific object registered in the Clearinghouse data base. |
| Add User | Creates a new user. You are prompted for the user's name (preferably a full name), a brief description (for example, the user's affiliation, office number, etc), an initial password, and one or more aliases. |
| Change Remark | Allows you to change the remark; that is, text description, of any object in the data base. NSMaintain prompts you for the object name, then for a new remark, offering the old remark as default. |
| Remove Alias | Removes an Alias from the data base. You are prompted for the alias. This has no affect on the primary object for which the removed name is an alias. |
| Remove User | Undoes the effect of Add User; that is, removes a user name and all its properties from the data base. |
| Remove Registered Object | Removes a specified object and all its properties from the data base. The primary name and description of the object are printed first and you are asked to confirm the deletion. You can use this to remove groups and other kinds of objects. |

## Manipulating Groups

A group is a Clearinghouse object with members. Groups are most commonly used for mailing lists and access control. The members can be either individuals or other groups. In the case of groups used for access control, a member can also be a pattern in which "*" usually replaces one or more entire fields of a three-part name.

A group has associated with it two access control lists: *owners* and *friends*. Owners can make any change to a group; they are like domain administrators for the narrow scope of the group itself. Friends are allowed to add or remove themselves from the group. For example, common interest groups typically have "open" membership, consisting of a friends list of "*:domain", or even "*:*:*" for a completely open group.

If the Owners or Friends list is empty, it defaults to the administrators of the domain. However, if the Owners list is non-empty, it *overrides* the administrators list. For example, if you remove yourself from the owners of a group, you can no longer modify the list, even though you are a domain administrator. The defaulting can lead to confusion, especially since the Describe command does not (and unfortunately cannot) indicate whether the owners and friends it displays are explicit or defaulted. For example, if a group previously had no explicit owners, then the Remove Owner command cannot be used, and any use of the Add Owner command implicitly removes all the domain administrators.

| | |
|---|---|
| Add Group | Creates a new user group. You are prompted for the group's name, a short description of the group, its initial members one at a time, its owners and its friends. NSMaintain checks all of the names except those that are patterns to ensure that you gave valid Clearinghouse names, and to resolve aliases. The Clearinghouse does not actually require that members of a |

group be registered Clearinghouse names, as it does not attach explicit meaning to the contents of a group until told to do so. Thus, if you type an invalid name, NSMaintain asks whether you really meant it, and keeps the name if you answer yes. Note, however, that any group used for access control must contain only registered Clearinghouse names or patterns.

If you specify any owners, you are always made an owner yourself as well, whether you explicitly said so or not, so as to avoid the anomaly of your not being able to further modify the group. You can, of course, remove yourself afterward if you really meant to.

| | |
|---|---|
| Add Member<br>Add Friend<br>Add Owner | Adds a member, friend or owner to a group. |
| Add Self | Adds you, the currently logged in user, to a group. You must be a friend or owner of the group. |
| Remove Member<br>Remove Friend<br>Remove Owner | Removes a specified member, friend or owner from a group. |
| Remove Self | Removes you, the currently logged in user, from a group. You must be a friend or owner of the group. |

### Manipulating Domains

These commands change the list of administrators of a domain. You must be an administrator of the domain or the parent organization to do this.

| | |
|---|---|
| Add Domain Administrator | Adds a user to the set of administrators for a domain. |
| Remove Domain Administrator | Removes a specified user from the administrators for a domain. |

## Errors

NSMaintain always ends each command with some sort of feedback about the completion of the operation. In information commands, the feedback is, of course, the requested information. In commands that change the data base, NSMaintain usually prints "done". If a command fails, NSMaintain prints a terse error message. Listed here are some of the more common ones:

| | |
|---|---|
| NoSuchObject | You asked about a name that does not exist in the Clearinghouse data base. Check that the spelling and the domain are correct. |
| IllegalOrganization<br>IllegalDomain<br>IllegalObject | The name you gave is not legal as a Clearinghouse name. Since NSMaintain already checks for incorrect use of asterisks, this usually means the name is too long. (The name component must be no more than 40 characters long; domains and organizations are limited to 20 characters each.) |
| Missing | The name you specified for a group, such as in the AddMember command, is not a group. |

| | |
|---|---|
| CredentialsInvalid | |
| VeriferInvalid | You are logged in incorrectly; that is, either your name or your password is incorrect. You can use the Change Login command to log in correctly. |
| AccessRightsInsufficient | You do not have the authority to make the change you requested. You can find out who does have the authority by using the command Describe for changing a group, or List Domain Administrators for all other changes. |
| NoChange | The change you requested would have no effect; for example, you added to a group a name that was already a member, or requested to remove a name that was not there. |
| TooBusy | The Clearinghouse contacted by NSMaintain was too busy to field the request. Lisp's present Clearinghouse implementation, unfortunately, does not handle this error, so passes it along to you. If you repeat the operation it may succeed. If this error persists for a long time, you may want to evaluate (START.CLEARINGHOUSE T) to completely clear the Clearinghouse cache; the system may then succeed in locating a more responsive server. |

## Examples

In the example session that follows, all user input is in boldface; everything else is typed by the system. To avoid clutter, carriage returns typed by the user are not shown. In many cases, a simple carriage return accepts the default input typed by the system, or completes a partially typed name. For clarity, most of the user input is in uppercase, although lowercase is equally acceptable. Commentary is in italics.

```
64> (NSMAINTAIN)
[Default login:  Arthur Dent:Research:ACME;
 Default domain: Research:ACME]
                                  NSMaintain shows me the defaults.
                                  (My password has not, however, been verified.)

CH: Describe name: EDISON:Research:ACME ...

Thomas A. Edison:Research:ACME is a User (Electronics Div., Rm 2732)
Aliases: Edison:, Wizard:         Domain and organization are elided here
Mailboxes: [Time:  8-Aug-86 17:30:54; Mail.Service: (Snail:)]
Userdata: [Last.Name.Index: 10; File.Service: Phylum:]
                                  The Userdata property is used by Viewpoint

CH: List Groups by pattern: *:Research:ACME ... AllResearch, Consultants,
ED, LispImplementors, LispInterest, NetAdministration, Skiiers, Staff,
WireBusters

CH: Show Details of previously listed names
```

```
  name: Consultants                  "C<cr>" is all that I typed

Consultants:Research:ACME is a User Group (Part-time personnel)
Owners: Staff:
Friends: NetAdministration:

  name: SKiiers

Skiiers:Research:ACME is a User Group (Snow sport enthusiasts)
Owners: UserAdministration:All Areas, Perry White:
Friends: *:*                         Anyone in organization ACME can join

  name:

CH: List Members of group: Skiiers:Research:ACME ...
Alexander G. Bell:Telcom, Christopher Craft:, Staff:

CH: Add Self to group: Skiiers:Research:ACME ...  done
                                     Skiiers was offered as default, being the last
                                     group I mentioned—I had only to type a cr.

CH: Add Self to group: Skiiers:Research:ACME ...  failed: NoChange
                                     I.e., I'm already a member

CH: List Servers of type FIle
 by pattern: *:Development:ACME ... Arrow, Quiver

CH: List Users by pattern: Ed*:Research:ACME ... (none)
                                     There are no users whose full name starts "Ed"

CH: List ALiases by pattern: Ed*:Research:ACME ... Edison, Educators
                                     But there are some aliases (not necessarily all users)

CH: List objects having property wORKSTATION
 by pattern: *M*:Research:ACME ... Archimedes, Camero, Cardamom,
Homestead, MayDay, Mendel, Ramanujan, SatanicMechanic, TheTajMahal

CH: Show Details of previously listed names
  name: Archimedes

Archimedes:Research:ACME is a Workstation (1186 in Rm. 2732)
Address.List: (6285#0.125101.20200#0)
Authentication.Level: [Simple: true; Strong: false]

CH: Change Default Domain (for name entry) to be: Development:ACME
Set this default globally as well (i.e. for use outside Maintain)? N

CH: Add Alias for object: Newton:Development:ACME
 Alias: Isaac:Development:ACME ...  done

CH: Describe name: Newton:Development:ACME ...

S. Isaac Newton:Development:ACME is a User (Apple Tester, Rm. 34)
Aliases: Isaac:Development, SIN:Development
Userdata: [Last.Name.Index: 0; File.Service: Arrow:Development]
```

```
CH: Remove Alias alias: SIN:Development:ACME ... done, alias was removed
from S. Isaac Newton:Development

CH: Add User
New user's name: Charles S. Brown:Development:ACME ...
Remark (terminate with CR): Test Team captain
Alias: Chuck:Development:ACME
Alias: Brown:Development:ACME
Alias: CSB:Development:ACME
Alias:  xxx                           Bare <cr> was typed to end the list
Initial password: ******* (retype password) *******...  done
                                      Chuck can later use Change Password to
                                      set a password of his own choosing.

CH: Add Group
New group name: Entomologists:Development:ACME ...
Remark (terminate with CR): Seekers of bugs

Enter names of members, owners and friends, one per line, terminated with
a blank line.

Member: brown:Development:ACME = Charles S. Brown:Development:ACME
                                      NSMaintain resolves alias, so that member
                                      names are in canonical form
Member: F. Kafka:Development:ACME
Member:  xxx

(If you enter no owners, the group will be owned by the administrators of
Development:ACME.)

Owner: brown:Development:ACME = Charles S. Brown:Development:ACME
Owner:  xxx

Friend: *:Development:ACME
Friend: *:Research:ACME
Friend:  xxx

Adding members... done
Adding owners... (including Arthur Dent:Research:ACME) done
                                      I'm an owner, too (else I couldn't modify the group)
Adding friends... done

CH: Remove User: BILBO:Development:ACME ...
Bilbo Baggins:Development:ACME (Furry ring finder)
 Confirm deletion (y or n): Y
 done

CH: Quit [confirm]                    Back to the exec now.
```

[This page intentionally left blank]

# PRESS

Xerox Lisp includes utilities for generating hardcopy in "Press" format. This is a file format for communicating documents to Xerox laser xerographic printers of the Press, Spruce or Fullpress class, which are known by the names Dover, Penguin, Raven and Spruce.

Note: Press, along with other classes of printers, is described fully in two sections in the *IRM*, "Hardcopy Facilities" and "Fonts."

The reason for putting this document into the *Lisp Library Modules Manual* is that support of these printers is no longer an integral part of the system.

## Requirements

FONTS.WIDTHS
PUPPRINT.LCOM
A Press print server

## Installation

Load PRESS.LCOM from the library.

Set the variable PRESSFONTWIDTHFILES (see below).

## User Interface

You can use the usual means:
HARDCOPY in background (right-button) menu,
HARDCOPY in FileBrowser menu, or
functions typed into the exec window (see below).

## Functions

| | |
|---|---|
| DEFAULTPRINTINGHOST | [Variable] |
| PRESSFONTWIDTHFILES | [Variable] |
| SEND.FILE.TO.PRINTER | [Function] |
| LISTFILES | [Function] |

(All: see the *IRM* for full details.)

## Limitations

PRESS does not support NS characters or NS fonts.

ReadNumber contains functions for implementing a calculator-type menu for entering numbers with the mouse.

## Installation

Load READNUMBER.LCOM from the library.

## Functions

ReadNumber functions are called either from the Executive window or programmatically from another process.

The numbers captured by ReadNumber are passed to whatever process currently has the TTY.

### Create a Key Pad

(RNUMBER *MSG POSITION MSGFONT DIGITFONT INCLUDEABORTFLG FLOATINGPTFLG POSITIVEONLYFLG ACCEPTTYPEINFLG* ) [Function]

Brings up a menu that looks like a ten-key calculator pad. Your selections, made by pressing the left mouse button when the cursor is on a digit, are accumulated in a displayed total. The key pad includes a backspace key (BS), a clear key (CLR), and a +/- key (-). When OK is selected, the total is returned.

If *MSG* is given, it is displayed at the top of the menu.

If *POSITION* is given, the menu will be put there; otherwise it will be put at the cursor.

If *MSGFONT* is given, *MSG* will be printed in it. If *MSGFONT* is NIL, DEFAULTFONT is used.

If *DIGITFONT* is given, the labels on the keys will be printed in that font. If *DIGITFONT* is NIL, BOLDFONT is used.

If *INCLUDEABORTFLG* is non-NIL, the menu will also include an abort key (abt). If the abort key is pressed, RNUMBER returns NIL.

If *FLOATINGPTFLG* is non-NIL, the menu will include a decimal point, and the value returned may be a floating point number.

If *POSITIVEONLYFLG* is non-NIL, the menu will not include a +/- key (-) and you will only be able to input positive numbers (but see *ACCEPTTYPEINFLG* ).

If *ACCEPTTYPEINFLG* is non-NIL, the menu will also respond to user-typed input (i.e., numbers typed in on the keyboard, rather than selected with the mouse). In this mode, carriage return corresponds to OK.

Note: The decimal point (.) and the minus sign (-) are also accepted, even though they are not options in the key pad menu.

If you close the key pad window, the action taken by RNUMBER depends upon the value of *INCLUDEABORTFLG*. If *INCLUDEABORTFLG* is NIL, RNUMBER generates an error (i.e., calls (ERROR!)). If *INCLUDEABORTFLG* is non-NIL, RNUMBER returns NIL (the same thing it does if the abort key is pressed).

## Create a Key Pad for Repeated Use

For some applications, it may be beneficial to avoid the creation of the key pad menu window each time a number is asked for. The following functions allow you to create a key pad menu window and use it repeatedly to get values from you.

Note: When used in this manner, a key pad menu window can only be used by one process at a time.

(CREATE.NUMBERPAD.READER *MSG WPOSITION MSGFONT DIGITFONT INCLUDEABORTFLG FLOATINGPTFLG POSITIVEONLYFLG*) [Function]

Creates a window suitable for use by NUMBERPAD.READ (see below). Its arguments are the same as for the function RNUMBER.

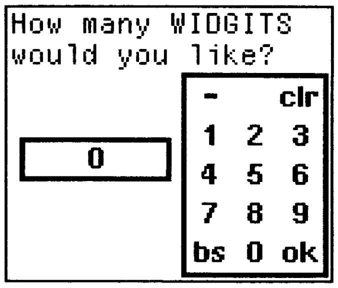
(NUMBERPAD.READ *NUMBERPAD/READER ACCEPTTYPEINFLG*) [Function]

*NUMBERPAD/READER* should be a window returned by the function CREATE.NUMBERPAD.READER (see above). NUMBERPAD.READ uses the window in the same manner as the function RNUMBER.

## Examples

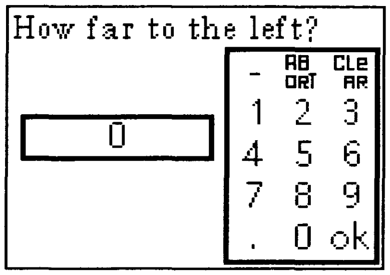
```
(RNUMBER "How many WIDGITS would you like?")
```

will result in the following menu being popped up:



```
(RNUMBER "How far to the left?") NIL '(CLASSIC
12) '(MODERN 14) T T)
```

will result in the following menu being popped up:



## Limitations

If you choose both FLOATNGPOINTFLG and INCLUDEABORTFLG, then there is no room for the backspace key, so the input is correctable only by selecting CLEAR and starting over. However, if ACCEPTTYPEINFLG is T, the keyboard's backspace key can be used.

[This page intentionally left blank]

The 1108 and 1186 each support two RS232 ports. One of these ports is configured as Data Terminal Equipment, and is intended to be connected to modems or terminal ports on other computers. (On the 1108, this port is available only with the addition of the E30 option.) The other port is configured as Data Communication Equipment, and is meant to drive printers or terminals. In this document, the DTE port is called the RS232 port, and the DCE port is called the TTY port.

Lisp provides a stream-oriented interface to the RS232 hardware. Users' programs can open streams to the hosts {RS232} or {TTY}, and perform input or output using standard Lisp I/O functions, such as READ-BYTE, READ-CHAR, etc.

Programs may use RS232 streams or TTY streams with the same programmatic interface; however, the RS232 port is preferred over the TTY if the application expects to handle large amounts of input data. In the 1108 and 1186, data entering the RS232 port is buffered independently of Lisp by the I/O processor (IOP). In addition, the RS232 software provides an additional layer of character buffering, freeing user programs from having to monitor the RS232 hardware frequently.

No independent buffering is provided for data entering the TTY port. As a result, Lisp cannot guarantee to catch all characters received on this port. For this reason, the TTY port should be used primarily to drive output devices such as printers.

## Requirements

### Hardware

To connect to a modem or another device, you need an RS232 or TTY cable (see the wiring diagrams in the Introduction of this manual).

For the 1108 only, you need the E-30 upgrade kit.

For the RS232 port to operate correctly, the remote device must assert the standard RS232 signals DSR and CTS. This requirement means that the appropriate pins on the 1186 be driven high, either properly by connecting the 1186 side to the corresponding device side, or permanently by jumpering the pins.

In order for the TTY port to operate correctly, the remote device must assert DTR and RTS. This requirement means that the appropriate pins on the 1186 be driven high, either properly by connecting the 1186 side to the corresponding device side, or permanently by jumpering the pins.

## Software

You need the following .LCOM files in order to run this module successfully:

DLRS232C or DLTTY (one of these is required)

RS232MENU

RS232CHAT or TTYCHAT, and KERMIT (these are optional).

(See also the file dependencies enumerated in the Introduction of this manual.)

## Installation

Load the required .LCOM modules from the library.

Run RS232C.INIT or TTY.INIT to set parameters.

## Using the RS232 Port

Support for the RS232 port is contained in the file DLRS232C.LCOM. Before using the RS232 port, it is necessary to initialize the RS232 hardware. The function RS232C.INIT is provided for this purpose:

(RS232C.INIT *BAUDRATE BITSPERSERIALCHAR PARITY NOOFSTOPBITS FLOWCONTROl*)

[Function]

The arguments correspond to the port parameters (see below).

Alternatively, the *BAUDRATE* argument can be an instance of the RS232C.INIT record. If *BAUDRATE* is NIL, the value of the global variable RS232C.DEFAULT.INIT.INFO is used in its place. This provides a means of automatically initializing the RS232 hardware without user intervention.

RS232C.DEFAULT.INIT.INFO [Variable]

This variable controls default initialization of the RS232 port. Its value may be set in the site INIT.LISP file, or in your INIT.LISP file. If RS232C.DEFAULT.INIT.INFO is not set when the RS232 module is loaded, its fields will be set to the following default values:

BaudRate: 1200
BitsPerSerialChar: 8
Parity: NONE
NoOfStopBits: 1
FlowControl: XOnXOff

Programs may use the Lisp function OPENSTREAM as an alternative to calling RS232C.INIT directly, with the parameters bundled up into the PARAMETERS argument.

For example, (RS232C.INIT 9600 8) can also be achieved by:

```
(OPENSTREAM '{RS232} 'INPUT NIL '((BaudRate
9600) (BitsPerSerialChar 8))
```

(RS232C.SET.PARAMETERS *PARAMETERLIST*) [Function]

This function allows applications to change the settings of the RS232 hardware while the RS232 port is in use.

*PARAMETERLIST* is an association list of parameter names and values. The following example sets the baud rate to 9,600 baud, and the character length to eight bits:

```
(RS232C.SET.PARAMETERS '((BaudRate . 9600)
(BitsPerSerialChar . 8)))
```

The following is a list of legal parameter names and values:

BaudRate 1186: 50, 75, 110, 150, 300, 600, 1200, 2400, 3600, 4800, 7200, 9600, 19200.
1108: all of the above, and 28800, 38400, 48000, 56000, 57600.

BitsPerSerialChar 5, 6, 7, 8 bits of data. If 5 or 6 bits of data are sent, they should be DATA, not CHARACTER.

Parity NONE, ODD, EVEN (1 parity bit).

NoOfStopBits 1, 1.5, 2 (i.e., the stop bit may have a period equal to 1, 1.5 or 2 bit-widths). not 1.5 on DL RS232

FlowControl NIL, NONE, XOnXOff, list.

NIL and NONE: no flow control.

XOnXOff: flow control using Xon and Xoff characters. For applications requiring XOn and XOff characters other than ↑Q and ↑S respectively, this parameter may be supplied as a list in the form: (1 <XOn> <XOff>), where <XOn> and <XOff> represent the character codes (ASCII 1 - 127) of the characters which are to be treated as the XOn and XOff characters. The leading "1" signifies that flow control should be enabled; a leading "0" will program the RS232 port with the appropriate XOn and XOff characters, but leave flow control disabled.

ModemControl This parameter should be a list of modem control signals to be enabled, such as DSR, CTS, DTR, DTR, RI and HS.

The functions RS232MODEMCONTROL, RS232MODEMSTATUSP, and RS232MODIFYMODEMCONTROL provide finer control over the settings of modem signals (see below).

DTR This parameter enables or disables the data terminal ready signal; it may be specified as T or NIL.

RTS This parameter enables or disables the request to send signal; it may be specified as T or NIL.

(RS232C.GET.PARAMETERS *PARAMETERLIST*) [Function]

The current settings for the RS232 port may be obtained at any time by calling this function.

*PARAMETERLIST* should be a list of parameter names. RS232C.GET.PARAMETERS returns an association list of parameter names and values, in a format acceptable to RS232C.SET.PARAMETERS.

(RS232C.SHUTDOWN) [Function]

The RS232 port is turned off by calling this function. It disables the RS232 port and closes any open streams on the devices.

## Using RS232 Streams

Programs may open streams to the RS232 port by calling OPENSTREAM with the file name {RS232}. The *ACCESS* argument to OPENSTREAM controls whether an INPUT or OUTPUT stream is returned. RS232 streams are unidirectional; to obtain a second stream open for the opposite access, call the function RS232C.OTHER.STREAM.

Only one pair of RS232 streams may be open at a time; an error will result if you attempt to open more.

(RS232C.OTHER.STREAM *STREAM*) [Function]

*STREAM* should be an RS232 stream. If *STREAM* is open for INPUT, an RS232 stream open for OUTPUT is returned; conversely, if *STREAM* is open for OUTPUT, an RS232 stream open for INPUT is returned.

The following Lisp functions are defined to work on RS232 streams open for the appropriate access: BIN, BOUT, READP, OPENP, CLOSEF, and FORCEOUTPUT.

(RS232C.CLOSE-STREAM *DIRECTION*) [Function]

This function closes one or both RS232 streams, so you don't need to have access to the streams to close them.

*DIRECTION* can be one of INPUT, OUTPUT, BOTH, or NIL. The function closes the RS232 stream open in *DIRECTION* mode; if *DIRECTION* is BOTH or NIL the input and output streams will be closed, if they exist.

RS232 streams are buffered. Input and output are performed in units of packets of data. The I/O processor collects incoming data into a packet, and makes that packet available to Lisp when one of the following conditions is true:

The packet is filled.
The frame time-out has expired.

The frame time-out is the number of hundredths of a second that are allowed to occur between the reception of single characters. This parameter is automatically set by the RS232 module. Its value depends on the baud rate of the RS232 port. If the value is set too large, interactive applications such as Chat will suffer from uneven typeout; if the value is too small, a larger number of shorter packets may be exchanged between Lisp and the I/O processor, resulting in increased processing overhead.

Lisp buffers data for output in packets of up to 578 characters. Output packets are sent to the RS232 port when one of the following conditions is true:

The current output packet is full.

The user program calls the FORCEOUTPUT function to force the current output packet to be sent.

The output stream is closed.

Applications that generate a large amount of output slowly may wish to reduce the size of outgoing packets. Although this will require additional processing overhead, it will cause output to occur more frequently without the program explicitly calling FORCEOUTPUT.

(RS232C.OUTPUT.PACKET.LENGTH *NEWVALUE*) [Function]

This function returns the current setting of the variable that controls the maximum size of output packets. If *NEWVALUE* is supplied, the setting is changed to *NEWVALUE*. *NEWVALUE* may be a number between 1 and 578.

Specifying a value is a matter of how long you are willing to wait at input time before you are assured of seeing incoming data. You can do a straightforward division to set the length (e.g., at 9600 baud, you get about 1 char/millisecond, so the max delay is 578ms; if you can tolerate only 1/4 second, then set it to 250 or so).

(RS232C.READP.EVENT *STREAM*) [Function]

Many RS232 applications are time-dependent. File transfer protocols such as Kermit and Modem depend on one or both sides of a file transfer detecting connection problems by means of time-outs. This function allows user programs to detect time-out conditions efficiently.

*STREAM* should be an RS232 stream open for input. This function returns an event that a program may wait upon for input data to become available on the stream. The following example illustrates how a program could wait up to 10 seconds for a character to become available:

```
[LAMBDA (STREAM)
    (LET ((EVENT (RS232C.READP.EVENT STREAM))
    (TIMER (SETUPTIMER 10000)))
    (until (OR (READP STREAM) (TIMEREXPIRED? TIMER))
    do (AWAIT.EVENT EVENT TIMER T)
    finally (RETURN (COND ((READP STREAM) (BIN STREAM]
```

## Using Modems

The following functions are useful for controlling modems:

(RS232SENDBREAK *EXTRALONG?*) [Function]

This function sends the out-of-band BREAK signal, for a period of 0.25 seconds; if *EXTRALONG?* is non-NIL, then the period is extended to 3.5 seconds.

(RS232MODEMCONTROL *SIGNALSONLST*) [Function]

This function is a Lambda-NoSpread function that sets the modem control lines to be "on" for the signals named in the list *SIGNALSONLST*; it returns the former setting of the lines. If *SIGNALSONLST* is not supplied (which is not the same as supplying NIL), then the control lines remain unchanged. The entries in *SIGNALSONLST* are symbol names for standard modem control lines; currently usable signal names are DTR and RTS.

(RS232MODIFYMODEMCONTROL *SIGNALSONLST SIGNALSOFFLST*) [Function]

Changes only those modem control lines specified in the union of the two arguments; those in *SIGNALSONLST* are set to be on, and those in *SIGNALSOFFLST* are set off. Returns the former state just as (RS232MODEMCONTROL) does.

(RS232MODEMSTATUSP *SPEC*) [Function]

Returns non-null if the reading of the modem status lines is consistent with the boolean form specified by *SPEC*; modem status signals currently supported are DSR, RI, and RLSD. *SPEC* may be any AND/OR/NOT combination over these signal names.

Example:

```
(RS232MODEMSTATUSP '(AND CTS (NOT RLSD))).
```

(RS232MODEMHANGUP) [Function]

This function takes whatever steps are appropriate to cause the modem to hang up. Generally, this means turning the DTR signal down for about three seconds, or until the DSR signal has gone down.

## Error Condition Reporting

The RS232 port detects parity errors, character framing errors, lost characters, and a number of other unusual conditions. As the I/O processor delivers each input packet to Lisp, it reports when the packet was received without error. If an error did occur while the packet was being received, Lisp will report this fact by writing a message to RS232C.ERROR.STREAM.

RS232C.ERROR.STREAM [Variable]

RS232 error conditions are reported on this stream. This stream is initially the PROMPTWINDOW.

(RS232C.REPORT.STATUS *NEWVALUE*) [Function]

There are circumstances in which the RS232 hardware believes it has encountered an error, when in fact it has not. A frequent cause is an incorrect parity setting in the RS232 port. Continually reporting RS232 errors is likely to slow RS232 processing severely. In cases where error reporting is not important, it is possible to disable error reports with the RS232C.REPORT.STATUS function. This function returns the current setting of status reporting, which may be one of the following:

| | |
|---|---|
| T | Errors are reported on both input and output. |
| NIL | Errors are never reported. |
| OUTPUT | Errors are reported on output only. |
| INPUT | Errors are reported on input only. |

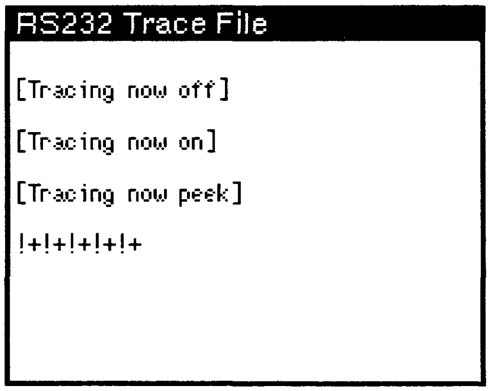
In addition, if *NEWVALUE* is supplied, the current setting of status reporting is changed to *NEWVALUE*.

## RS232TRACE

To help in debugging RS232 applications, it is possible to trace the data that is being sent out via the port. RS232 packets are traced using:

(RS232C.TRACE *MODE*) [Function]

*MODE* is one of PEEK, T, or NIL. If MODE is either T or PEEK, RS232C.TRACE will open a trace window in the mode selected. T indicates a full trace, with every byte being shown. PEEK is a less verbose trace, with every incoming packet shown as a " + " and every outgoing packet shown as a "!". NIL will turn off the tracing. Clicking the left mouse button in the trace window will cycle between the modes.



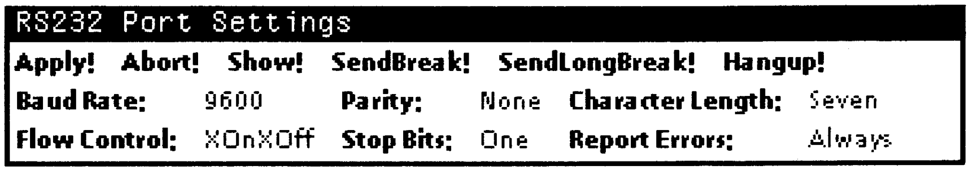
```
[Tracing now off]
[Tracing now on]
[Tracing now peek]
!+!+!+!+!+
```

## RS232CHAT

The RS232CHAT module is a facility that permits the Chat library module to communicate over the RS232 port. Once it is loaded, you may chat to the host named RS232 to open a connection to the RS232 port.

## RS232CMENU

RS232CMENU is a utility that provides a menu interface to controlling the settings of the RS232 port. When loaded along with RS232CHAT, this utility may be invoked by choosing the "Set Line Parameters" entry in the options menu that appears if you select "Options" in the middle-button Chat menu.


RS232 Port Settings

Apply! Abort! Show! SendBreak! SendLongBreak! Hangup!

Baud Rate: 9600 Parity: None Character Length: Seven

Flow Control: XOnXOff Stop Bits: One Report Errors: Always

The following commands are available in the RS232 menu:

| | |
|---|---|
| Apply | This menu item changes the RS232 port settings according to the values of the fields in the rest of the menu. No changes are made to the RS232 hardware until this command is given. |
| Abort | Closes the RS232 menu and aborts any changes that could have been made. |
| SendBreak | Sends a normal (0.25 second) break signal. Requires confirmation, as this command could cause the modem connection to be broken. |
| SendLongBreak | Sends a long (3.5 second) break signal. As above, this requires confirmation. |
| Hangup | Tries to hang up the modem (by dropping DTR). Requires confirmation. |

The following fields are multiple choice items; when their labels are selected with the mouse, a menu of possible values for the field appears. Choosing a value from the menu will change the field; clicking off the menu will leave the field unchanged.

| | |
|---|---|
| Baud Rate | Changes the BaudRate of the RS232 port. |
| Parity | Changes the Parity setting of the RS232 port. |
| Character Length | Changes the BitsPerSerialChar setting of the RS232 port. |
| Flow Control | Changes the FlowControl setting of the RS232 port. |
| Stop Bits | Changes the NoOfStopBits setting of the RS232 port. |
| Report Errors | Changes the RS232C.REPORT.STATUS setting of the RS232 port. |

## File Transfer Using RS232

Files may be transferred using RS232CHAT and either the Kermit or Modem protocols.

## Using the TTY Port

Support for the TTY port is contained on the file DLTTY.LCOM. The TTY port is designed to support low-speed communications with RS232 devices. The I/O processor offers no low-level support for input buffering. As a result, it is quite possible for newly received input characters to overwrite previously received but unread characters in the input hardware. The TTY port provides exactly one character's worth of buffering; each character must be read by Lisp before the next character is completely received.

Note: No hardware flow control is provided on the TTY port. The Lisp TTY port service routines will obey received flow control commands, but will not generate flow control commands in response to increased input data rate.

(TTY.INIT *BAUDRATE BITSPERSERIALCHAR PARITY NOOFSTOPBITS FLOWCONTROL*) [Function]

This function is very similar to the function RS232C.INIT.

Before using the TTY port, the TTY hardware must be programmed with the proper characteristics for your application.

Alternatively, the *BAUDRATE* argument can be an instance of the RS232C.INIT record. If *BAUDRATE* is NIL, the value of the global variable TTY.DEFAULT.INIT.INFO is used in its place. This provides a means of automatically initializing the TTY port hardware without user intervention.

TTY.DEFAULT.INIT.INFO [Variable]

This variable controls default initialization of the TTY port. Its value may be set in the site INIT.LISP file, or in your INIT.LISP file. If TTY.DEFAULT.INIT.INFO is not set when the TTY package is loaded, its fields will be set to the following default values:

BaudRate: 1200
BitsPerSerialChar: 8
Parity: NONE
NoOfStopBits: 1
FlowControl: XOnXOff

Programs may use OPENSTREAM as an alternative to calling TTY.INIT directly, with the parameters bundled up into the PARAMETERS argument as shown below.

For example:

```
(OPENSTREAM '{TTY} 'BOTH NIL '((BaudRate 9600)
(BitsPerSerialChar 8))
```

(TTY.SET.PARAMETERS *PARAMETERLIST*) [Function]

Applications may change the settings of the TTY hardware while the TTY port is in use.

*PARAMETERLIST* is an association list of parameter names and values. For example, let's set the baud rate to 9600 baud and the character length to eight bits:

```
(TTY.SET.PARAMETERS '((BaudRate . 9600)
(BitsPerSerialChar . 8)))
```

The following is a list of legal parameter names and values:

| | |
|---|---|
| BaudRate | 50, 75, 110, 150, 300, 600, 1200, 2400, 3600, 4800, 7200, 9600, 19200. |
| BitsPerSerialChar | 5, 6, 7, 8. If 5 or 6 bits of data are sent, they should be DATA, not CHARACTERS. |
| Parity | NONE, ODD, EVEN ( 1 parity bit). |
| NoOfStopBits | 1, 2 (This parameter should be 1 except at 110 baud). |
| FlowControl | NIL, XOnXOff. For applications requiring XOn and XOff characters other than ↑Q and ↑S respectively, this parameter may be supplied as a list in the form:(1 <XOn> <XOff >), where <XOn> and <XOff> are replaced by the character values of the XOn and XOff characters. The leading 1 signifies that flow control should be enabled; a leading 0 will program the TTY port with the appropriate XOn and XOff characters, but leave flow control disabled. Note: XOnXOff flow control is known to be reliable only up to 4800 baud. |
| DSR | This parameter enables or disables the data set ready signal; it may be specified as T or NIL. |
| CTS | This parameter enables or disables the clear to send signal; it may be specified as T or NIL. |

(TTY.GET.PARAMETERS *PARAMETERLIST*) [Function]

The current settings for the TTY port may be obtained at any time by calling this function.

*PARAMETERLIST* should be a list of parameter names. TTY.GET.PARAMETERS returns an association list of parameter names and values, in a format acceptable to TTY.SET.PARAMETERS.

(TTY.SHUTDOWN) [Function]

This function turns off (disables) the TTY port and closes any open streams on the device.

## Using TTY Streams

Programs may open streams to the TTY port by calling OPENSTREAM with the file name {TTY}. The ACCESS argument to OPENSTREAM may be INPUT, OUTPUT, or BOTH.

Unlike RS232 streams, TTY port streams are not buffered, and a single stream may be used for both input and output. The generic Lisp input/output functions BIN, BOUT, READP, OPENP, and CLOSEF may be used on TTY port streams.

### TTYCHAT

TTYCHAT is a module that enables the Chat library module to communicate over the TTY port. No user-callable functions or user-settable variables are available in the TTYCHAT module. Once it is loaded, you may chat to the host named TTY to open a connection to the TTY port. TTYCHAT is contained on the file TTYCHAT.LCOM.

Because the TTY port does no low-level input buffering, it is quite likely that many input characters will be lost while chatting at 1200 baud or higher. This package should only be used for non-critical applications, such as testing connections between the TTY port and low-speed printers.

### RS232CMENU

RS232CMENU is compatible with the TTY port as well. Certain RS232 port commands, such as SendBreak! are not available with the TTY port, and hence do not appear in the menu.

## Examples

### Testing the Connection Between Two Xerox Lisp Machines

To test for a working RS232 connection between Machine A (a Xerox 1108) and Machine B (a Xerox 1100, 1108, or 1186) by moving a file between them, proceed as follows:

Load RS232CHAT.LCOM on both machines.

Call RS232C.INIT to set up parameters.

Do RS232CHAT on both machines. You will be prompted for a window.

Whatever you type on machine A should be echoed on machine B and vice versa.

### Testing the Connection Between Xerox Lisp Machines and a VAX Running VMS

VAX side: Set baud rate at which files will be transferred on the VAX side using the VMS command SET. For example, to set to 1200 baud, type:

```
SET TERM TTA1:/SPEED=1200/PERM
```

1108 side: Load RS232CHAT.LCOM

Initialize: `(RS232C.INIT 1200 8 NIL 1 NIL'RS232C)`

Then call `(RS232CHAT)`

You should be able to use the Chat window like a VAX teletype terminal.

By matching the baud rate on the VAX ( through SET TERM) with that on the 1108 (through RS232C.INIT), you can use any speed up to 9600 baud.

# SAMEDIR

SameDir modifies MAKEFILE to guard against inadvertently writing out a file onto a directory other than the one it came from.

SameDir adds the form (CHECKSAMEDIR) to MAKEFILEFORMS. It compares the (HOST&DIRECTORYFIELD *OLDFILE*) against (DIRECTORYNAME T T) to see whether the connected directory matches the old file's source.

## Installation

Load SAMEDIR.LCOM from the library.

## User Interface

If you do a MAKEFILE and you are connected to a directory that is not listed in the FILEDATES property of the file, and the file has a FILEDATES property at all (i.e., this isn't a brand new file), the system will prompt you with:

**You haven't loaded or written TORTOISE in your connected directory {server}<user> should I write it out anyway?**

Your options are reply with Y, N, C, or O:

Y Yes, do the MAKEFILE

N No, abort the MAKEFILE

C Connect to other directory: allows you to type in another path.

O Oops! Connect to the best guess; i.e., the directory where the file was last loaded or written. This option requires confirmation, in case you don't like the directory that the system prompts you with.

The default answer to the question is Y (do the MAKEFILE).

When comparing directory names, SameDir ignores case differences between the old and new directory names.

MIGRATIONS [Variable]

For those who regularly LOADFROM files on one directory and MAKEFILE elsewhere, the variable MIGRATIONS can be set to keep SameDir from asking too often. It is an association list containing pairs of (*OLDDIR* . *NEWDIR*), which specifies which migrations are allowable.

For example, if it is legitimate to LOADFROM a file on {MYHOST}<PUBLIC> and then do a MAKEFILE to {MYHOST}<TEST>, then adding ({MYHOST}<PUBLIC> . {MYHOST}<TEST>) to MIGRATIONS will prevent MAKEFILE from complaining about such movement.

## Limitations

For Unix hosts using the PUP FTP protocol, there is sometimes an inconsistency between the directory name in the full file name and the directory name in DIRECTORYNAME. SameDir may have trouble in that case detecting that the directories are the same.

# SPY

Spy is a tool to help you make programs run faster by giving you a picture of where the program is spending its time.

## Description

Spy has two parts: a sampler and a displayer. The sampler runs while your program is running, and it monitors what your program is doing. The displayer displays the data gathered by the sampler.

The sampler periodically interrupts the running program to account the functions in the current call stack. This allows Spy to remember not only (proportionally) how long is spent in each individual function, but also how long each function is seen on the call stack. The sampler data structures minimize interference with the normal running of the program — there is little noticeable performance degradation. Spy doesn't log every call and return (it only samples), so you can run it even over long computations without fear of overflowing storage limits.

The displayer uses the Grapher module to display the data gathered by the sampler. In the graph, the height of each node is adjusted to be proportional to the amount of time. Just as MasterScope and Browser give an interactive picture of the static structure of the program, Spy gives an interactive picture of the dynamic structure. The displayer is interactive as well as graphic. That is, you can look at the data in a variety of ways, since it seems there is no one picture that says it all. Since the displayer knows the whole call graph, it can show the entire tree structure, with separate calls to a function accounted separately, or merge separate calls to the same functions. Since the sampler records the entire calling stack when it samples, it can account for both individual and cumulative time. When the sampler runs, if a function is on the top of the stack, it adds to its individual total; if the function is on the stack at all, the sampler adds to the cumulative total.

When there are several calls to the same function within the graph, the displayer can either merge the nodes (show the total time for the function in one node) or not. If a node is merged, then one of the boxes in the graph will have all of the time for that function accounted to it, and the rest will be left as ghost boxes. Spy has a variety of ways of controlling which nodes will be merged.

## Requirements

GRAPHER
READNUMBER
IMAGEOBJ

## Installation

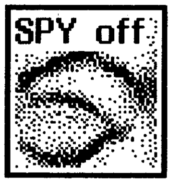
Load SPY.LCOM and the required .LCOM modules from the library.

## User Interface

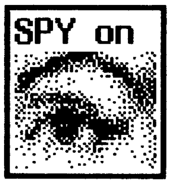
(SPY.BUTTON *POS*) [Function]

This function puts up a little window, which you can use to turn Spy on and off. If *POS* is NIL, you can drag the window with the mouse. If *POS* is specified (in the format xxx . yyy) then the window is placed at those coordinates.

When Spy isn't watching, it looks like this:



Left-clicking (pressing the left mouse button) on it once will turn on sampling, and the window will look like this:



Clicking on it again will turn off sampling, and display the results (SPY.TREE 10). This is the simplest way of spying on operations. (See SPY.TREE below.)

Note: You can't turn off sampling if the mouse process is locked out.

(SPY.START) [Function]

Reinitializes the internal Spy data structures, and turns on sampling.

(SPY.END) [Function]

Turns off sampling, and cleans up the data structures in preparation for the display phase performed by SPY.TREE.

(SPY.TOGGLE) [Function]

If Spying is off, turn it on with (SPY.START). If it is on, turn it off with (SPY.END) and then show the results with (SPY.TREE 10).

It is reasonable to use this with an interrupt character; e.g.,

```
(INTERRUPTCHAR (CHARCODE ↑C) '(SPY.TOGGLE) T)
```

will enable control-C (or any other character you specify) as an interrupt which will turn spying on and off, the same as clicking on the Spy button. (If the Spy button is visible, it will respond to the interrupt.) Then, a control-C will turn on spying, and another one will turn it off.

(WITH.SPY *FORM*) [Macro]

Calls (SPY.START), evaluates *FORM*, calls (SPY.END), and then returns the value of *FORM*.

For example,

```
(WITH.SPY (LOAD 'FOO))
```

is basically

```
(PROGN (SPY.START) (PROG1 (LOAD 'FOO) (SPY.END].
```

(SPY.TREE *THRESHOLD INDIVIDUALP MERGETYPE DEPTHLIMIT*) [Function]

SPY.TREE displays the results of the last SPY sampling in a Grapher window. There are a number of parameters that control the display, which you can either set when you call SPY.TREE, or set interactively with the menu. You normally just use (SPY.TREE) and the menus.

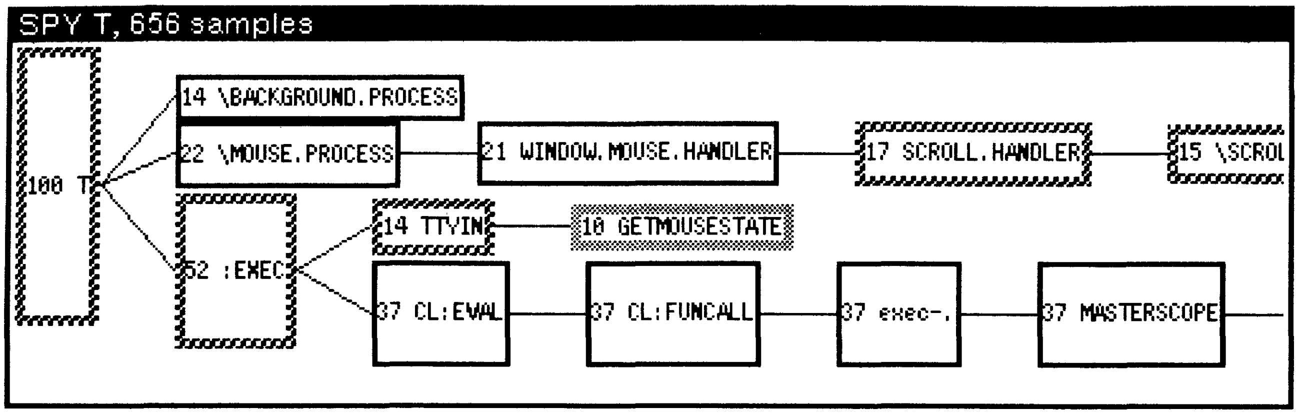
*THRESHOLD* is a percentage (defaults to 0). If a function's contribution to the total elapsed time is lower than the threshold percentage, it is not displayed.

*INDIVIDUALP* is either NIL or T. This controls whether cumulative or individual percentages are displayed. The default is cumulative, in which case a function is charged for the time spent in that function and all subfunctions; if individual, a function is only charged for the time spent in that function alone.

*MERGETYPE* is one of (NONE, ALL, DEFAULT). This controls accounting for functions that appear in several places in the calling tree. Mergetype ALL indicates the total time spent for all calls to the same function, regardless of where it appears. Mergetype NONE indicates the times separately for each instance of the function. Mergetype DEFAULT is the same as NONE except for recursive functions.

*DEPTHLIMIT* is a number (defaults to NIL = arbitrary depth; not completely debugged for other values).

You will get a prompt to open a window, and then a graph will appear in it, something like this:



In this example, 100% of the time was spent under the top frame, T. That time was then divided up among three processes: \BACKGROUND.PROCESS, \MOUSE.PROCESS, :EXEC. The numbers to the left of the label are the percentages. The height of the box is proportional to the percentage (except that it is always made big enough to hold the label). The width isn't significant; it is just wide enough to hold the name of the function

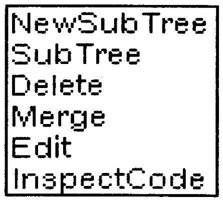
You can point at any of the nodes.

## Right Button Operation

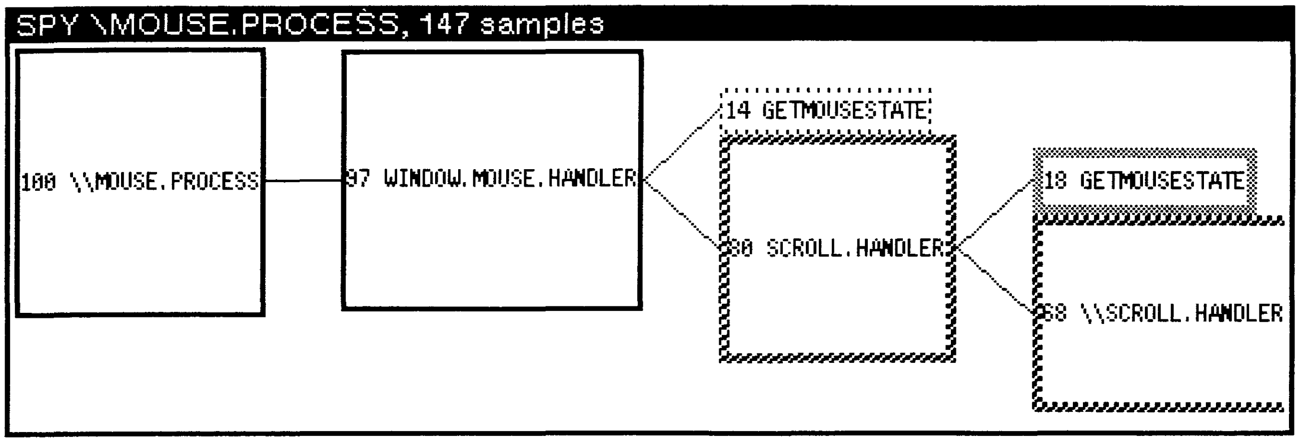
If you right-click on a node, the window title will change to show the function name, and the individual and cumulative percentages. The first number is the individual total and the second is the cumulative. If the right-click is not on a node, the title will change to show the name of the top frame and the total number of samples.

## Left/Middle Button Operations — on a Node

If you left-click, or middle-click on a node, you will get a menu:



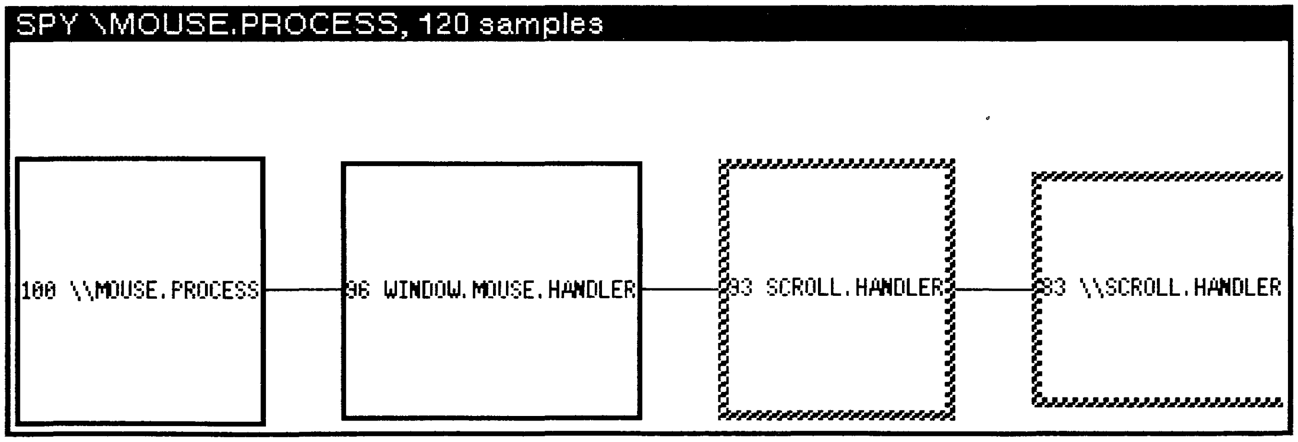
NewSubTree  Creates another Spy window that includes data only from this node and its descendents (with the tree rooted at the selected node). Suppose that you were only interested in the actions that you had invoked with the mouse. You can left-click \MOUSE.PROCESS and select the NewSubTree option. You then get a picture like this:



SubTree — Behaves just like NewSubTree except that Spy will reuse the same window.

Delete — Removes the selected item (and its subbranches) from consideration in this window, and redisplays.

For example, if you don't want to consider GETMOUSESTATE in the graph at all, you can delete the GETMOUSESTATE box and get:



The percentage for the SCROLL.HANDLER changed from 80% to 93% when GETMOUSESTATE was deleted.

Merge — Allows you to merge a node with its caller everywhere in the tree.

Edit — Invokes SEdit for the function in the node.

InspectCode — Shows the compiled code for the function in the node.

## Left/Middle Button Operation — Not on a Node

If you press the left, or middle button while not on a node, you get a menu that will let you view or change the parameters for the Spy window:

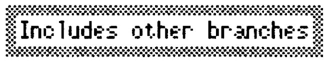
| | |
|---|---|
| Legend | Displays SPY border interpretation. |
| Inspect | Allows for the inspection of the current parameter settings for the Spy window. |
| SetThreshold | Sets the threshold for displaying a node. Any node whose percentage is below the threshold won't show up (unless it is needed to connect the graph together). You can set the initial threshold via the threshold argument to SPY.TREE; otherwise it defaults to zero. |
| Individual/Cumulative | This is used to toggle between the display of individual and cumulative times. The initial default is cumulative; you can set it to Individual by supplying T as the INDIVIDUALP argument to SPY.TREE. (See below). |
| MergeNone/MergeAll/MergeDefault | This controls merging of nodes; see below. |

## Ctrl-Left/Ctrl-Middle Button Operations

If you press the left, or middle button while the control key is down, you will get the same menu, but the action will be deferred until you next use the left/middle button. For example, you can delete several nodes and then do one update.

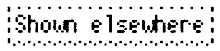
## Merged Nodes

If two nodes are merged, then the merged node will include the time (and descendents) of the other. The display of a merged node is different; it is shown with a thick gray border; e.g. ,



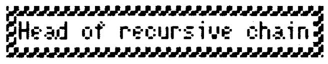
Includes other branches

The time in a merged node is the sum of the times for all occurrences. Other calls to the same function may show up as ghost boxes; e.g.



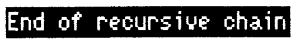
Shown elsewhere

The case where a function is merged with a recursive call to itself is handled specially: the head of the recursion is marked with a wider checkered border:



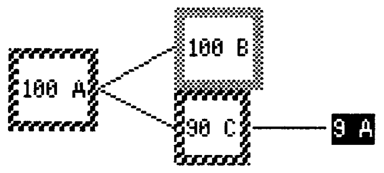
Head of recursive chain

and the tail of the recursion is shown in reverse video:



End of recursive chain

In the recursive case, you can find a situation like this:



In this case, A calls B and C, and C calls A. All of the time is really spent in B, although only 10% is due to the call from the top-level, and 90% is under a call to C, which called A, which called B. In this situation, C is also recursive, of course, and also has the recursive border. If you find this display confusing, try the MERGENONE option and see if you get a clearer picture.

MERGEDEFAULT means to merge any function that is not in SPY.NOMERGEFNS, initially set to (SI::*UNWIND-PROTECT* CL:EVAL \\EVAL-PROGN \\INTERPRET-ARGUMENTS \\INTERPRETER \\INTERPRETER1 ERRORSET \\EVAL \\EVALFORM APPLY \\PROGV EVAL).

MERGENONE means not to merge at all.

MERGEALL means to merge any two nodes for the same function.

The default for Individual mode is MERGEALL. The default for Cumulative mode is MERGEDEFAULT.

## Individual and Cumulative Modes

Spy initially comes up with the height of the boxes showing the amount of time the function was on the current call stack. This is called cumulative mode, since each function gets the time that both it and the functions it calls account for. There is another kind of display, called Individual, in which the boxes are proportional to the amount of time the function was on the top of the stack.

One thing to watch for: when you switch between Individual and Cumulative modes, the threshold stays the same. Sometimes the threshold for Individual needs to be higher; otherwise, functions will tend to disappear in the Individual tree. Also, switching to Individual mode also changes to MERGEALL, while switching to Cumulative changes you to MERGEDEFAULT.

(SPY.LEGEND) [Function]

If you forget what the different shadings and borders mean, this function brings up a window that shows what they mean; i.e., it shows the interpretation of SPY.BORDERS or the other internal controls.

SPY.FREQUENCY [Variable]

How many times per second to sample? Initially set to 10. (Maximum 60).

SPY.NOMERGEFNS [Variable]

Functions on this list won't get merged under MergeDefault. Includes (SI::*UNWIND-PROTECT* CL:EVAL \\EVAL-PROGN \\INTERPRET-ARGUMENTS \\INTERPRETER \\INTERPRETER1 ERRORSET \\EVAL \\EVALFORM APPLY \\PROGV EVAL). You may need to add more.

SPY.TREE [Variable]

This variable (same name as the function) is used to hold the data from the last sampling. You can save it and restore it using UGLYVARS (see *IRM*).

SPY.BORDERS [Variable]

Used to control the border display on a tree. This is a list of (*NODETYPE DESCRIPTION BORDERWIDTH TEXTURE INTERIORTEXTURE)*.

SPY.FONT [Variable]

Font used to display node labels. Initially GACHA 10.

SPY.MAXLINES [Variable]

Maximum height of a node in the graph, measured in multiples of the font height of SPY.FONT.

## Limitations

Spy doesn't know anything about the interpreter or the internal workings of Lisp. Internal functions that are not REALFRAMEP and don't normally show up on BT backtraces (but do on BT!) will be shown in Spy. This includes things like \INTERPRETER1, which will appear underneath any interpreted function call. Thus Spy does not distinguish between frames that are interesting or not interesting to the user.

# SYSEDIT

There are many system-internal data type declarations that don't usually appear in the sysout. For example, most people don't use the internal fields of strings or streams, so their definitions aren't included in the sysout. However, there is a collection of definitions that are used by people who are working on system-level code, definitions that are widely relied upon by more than one system source file.

SysEdit is provided for users who work with the system sources or who write system-level code — it brings in the frequently-used definitions.

## Requirements

MASTERSCOPE
EXPORTS.ALL
CMLARRAY-SUPPORT

## Installation

Lisp programming environment.

## Definitions

The declarations and definitions are provided in the source listings of SysEdit and in the files it loads.

## Limitations

SEDit internal definitions are not included, nor are declarations that are used only within a single system source file.

[This page intentionally left blank]

# TABLEBROWSER

TableBrowser implements a simple mechanism for building applications that browse certain kinds of tabular data. It supplies a set of basic functions that maintain the window, allowing scrolling and selection of items; the application defines the items, how they print, and any higher-level operations to perform on them. FileBrowser is an example of an application built upon TableBrowser.

## Installation

Load TABLEBROWSER.LCOM from the library.

During program development, you also need to load the declarations file TABLEBROWSERDECLS; this file is not needed when running a compiled application. The file manager coms for the typical application file should thus include the two commands

```
(FILES (SYSLOAD) TABLEBROWSER)
(DECLARE: EVAL@COMPILE DONTCOPY
     (FILES (SOURCE) TABLEBROWSERDECLS)
```

## User Interface

An instance of a TableBrowser application is a window displaying a browser. The browser consists of an ordered set of items. Each item contains an item of application data and some bookkeeping information. TableBrowser maintains the display, supplying generic scrolling, painting, reshaping and selection mechanisms; the application supplies TableBrowser with a set of methods for displaying the contents of an item, what to do when an item is copy selected, etc.

Review the user interface of the FileBrowser module to get an idea of the selection mechanism and the sort of functionality available from TableBrowser. Briefly, items are selected by clicking in the browser window — left to select a single item, middle to add an item to the current selection, control-middle to remove an item, right to extend the selection, control-right to extend the selection, including deleted items.

The typical application creates a browser window, fills it with items, and attaches one or more menus to the window. The menu contains commands that may perform application-specific operations, usually on the set of currently selected items.

## Records

There are two important records, TABLEITEM and TABLEBROWSER. These records are declared in the file TABLEBROWSERDECLS.

### TABLEITEM Record

Each piece of application data (a line in the browser) is encapsulated in an instance of the datatype TABLEITEM, whose fields are as follows. The first four fields are supplied by the application; the others are maintained by TableBrowser and should be considered read-only from the point of view of the application:

TIDATA [Record field]

An arbitrary pointer to the application data.

TIUNSELECTABLE [Record field]

If T, the user is not permitted to select this item.

TIUNCOPYSELECTABLE [Record field]

If T, the user is not permitted to copy-select this item.

TIUNDELETABLE [Record field]

If T, the user is not permitted to delete this item. This field is not currently supported by TableBrowser, though the application may use it itself for this purpose.

TI# [Record field]

An integer denoting the position of this among the set of items in the browser. This field is maintained automatically by TableBrowser as items are removed or new items are inserted. The first item is numbered 1.

TISELECTED [Record field]

T if the item is currently selected; NIL otherwise.

TIDELETED [Record field]

T if the item is currently marked for deletion; NIL otherwise.

### TABLEBROWSER Record

For each browser, there is an instance of the datatype TABLEBROWSER, whose fields are described below. TABLEBROWSERs are created by TB.MAKE.BROWSER (see the section "Functions," below). The first six fields listed here are "methods"— application functions called from the TableBrowser code when some event occurs or in order to carry out some operation. The application function is called with the browser object itself as the first argument, and possibly other arguments. Only the first method, TBPRINTFN, is required; the others may be NIL. For most applications, the fields listed here are set in the call to TB.MAKE.BROWSER and then never changed.

TBPRINTFN [Record field]

Application function invoked to print a single item in the browser. The three arguments are the browser, the item being painted, and the window in which the item is to be painted. At the time the method is called, TableBrowser has set the x,y position of the window to the left edge of the line where this item starts. It has done nothing to the line, including clear it; the application must clear the line before printing if it so desires. Somewhat smoother, but more complex, repainting is possible if the application clears only the areas that it is not otherwise overwriting. After the printing is finished, TableBrowser supplies a selection mark and deletion line if appropriate. TableBrowser worries about scrolling, reshaping, inserting new items, etc., and calls the printing function for each line that it has determined needs to be (re)displayed as the result of operations performed on the browser.

TBCOPYFN [Record field]

Application function invoked when an item in the browser is copy-selected. Copy selection occurs when the Copy or Shift key is held down and the mouse is used to select a line in the browser. While the Copy key is down, TableBrowser highlights the item under the cursor with a dotted underline; when the copy key is released, the highlighting is removed, and the copy function is called with two arguments: the browser and the item selected. The Copy function typically calls BKSYSBUF on some string or structure representing the item selected. Copy selection can be aborted by moving the mouse outside the window with the mouse button still down. If an item's TIUNCOPYSELECTABLE field is true, copy selection is ignored while the mouse is inside the item. If TBCOPYFN is NIL, then copy selection is ignored for the entire browser.

TBCLOSEFN [Record field]

Application function invoked when the user tries to close or shrink the browser window. The three arguments are the browser, the window, and a flag whose value is one of the symbols CLOSE or SHRINK. If the function returns the symbol DON'T, then the close or shrink operation is aborted; if it returns NIL, it proceeds; otherwise, the value is a function to run as a separate cleanup process. In this last case, the window remains open and the value returned from the TBCLOSEFN application is called in a new process with the same three arguments (browser, window, flag). When this function finishes, it should call the function TB.FINISH.CLOSE (see the section "Functions," below) to complete the close or shrink operation, assuming it wishes it to proceed.

TBAFTERCLOSEFN [Record field]

Application function invoked when the browser window is about to be discarded. The two arguments are the browser and the window. This function is called *after* the TBCLOSEFN, if any, has permitted the window to close; note that it is not called when a window is merely shrunk. The application might, for example, want to remove the browser from its own structures, snap circular links, etc. The return value is ignored.

TBTITLEEVENTFN [Record field]

Application function invoked when the middle mouse button is pressed while the cursor is in the title bar of the browser (clicking in the body of the window is for selection and is handled by TableBrowser itself). The two arguments are the window and the browser. Note that this is the only method that does not take the browser as the first argument; this is to match the form of window system button event functions.

TBFONTCHANGEFN [Record field]

Application function invoked when the font of the window is changed (by TB.SET.FONT). The two arguments are the browser and the window. The application function might, for example, want to change cached information about the size of the font.

TBLINETHICKNESS [Record field]

The thickness of the horizontal lines drawn through deleted items. This defaults to the value of TB.DELETEDLINEHEIGHT, initially 1. For example, setting this field to the height of an item would result in items being completely blacked out when they are deleted.

TBHEADINGWINDOW [Record field]

An optional auxiliary window that is to be horizontally scrolled in parallel with the main window. The WIDTH of the window's EXTENT property is maintained in synch with that of the main window, and whenever the main window is horizontally scrolled, the heading window is scrolled by the same amount. You still need to create this auxiliary window, attach it where you want it and supply it with a REPAINTFN. (This is how FileBrowser implements its header line identifying the columns of attributes.)

TBUSERDATA [Record field]

An arbitrary pointer to application-dependent data.

The following fields describe the size and shape of items. The values are usually derived from information supplied when a browser is created (see TB.MAKE.BROWSER) and then never changed. If an application wishes to change any of these fields once a browser has been created, it should call TB.SET.FONT afterwards to notify TableBrowser of the change.

TBFONT [Record field]

A font descriptor, the default font in which items are painted.

TBFONTHEIGHT [Record field]

The height of the font.

TB#LINESPERITEM [Record field]

The number of lines (in units of the font's height) occupied by each item. TableBrowser requires that each item occupy the same number of lines. The default is 1. For multi-line items, the window is positioned at the first line when its print function is called, selection markers point at the first line, and deletion lines are drawn only through the first line.

TBITEMHEIGHT [Record field]

The total height of an item. This is normally the font height times the number of lines per item, but an application can set it explicitly, independent of the font, instead of specifying the number of lines per item.

TBBASELINE [Record field]

The distance of an item's baseline above the bottom of the item. This field has been renamed from TBFONTDESCENT. This field is used for two purposes:

- When the browser's PRINTFN is called, the y-position of the window is set to be at the baseline.
- Selection marks and deletion lines are centered between the baseline and the top of the item.

The following fields are maintained by TableBrowser, but may be of use to application code; they should be considered read-only.

TBWINDOW [Record field]

A pointer to the window containing the browser.

TBLOCK [Record field]

A monitor lock acquired by the TableBrowser when performing operations on the browser. Application code may want to hold this lock while performing a series of TableBrowser operations that it wishes to have occur atomically. Selection and scrolling are inhibited while this lock is busy.

## Functions

This section describes the functions that create and manipulate TableBrowser windows and their contents. Typical use for most of these functions is from the code invoked by commands from a menu attached to the window by the application.

### Creating a TableBrowser

(TB.MAKE.BROWSER *ITEMS WINDOWSPEC PROPS*) [Function]

Creates a new browser, browsing *ITEMS*, a list of TABLEITEM records. *ITEMS* may be NIL, in which case an empty browser is created. It is permissible for the TISELECTED and/or TIDELETED fields to be true in any item; this is a way of setting the initial selection, and/or pre-deleting some items.

If *WINDOWSPEC* is supplied, it is a window or region; otherwise, the user is prompted for a window.

*PROPS* is a list in property list format specifying initial values for some features of the browser. The properties PRINTFN, COPYFN, CLOSEFN, AFTERCLOSEFN, LINESPERITEM, ITEMHEIGHT, BASELINE, HEADINGWINDOW, LINETHICKNESS, and USERDATA set the corresponding fields of the new TABLEBROWSER record (see above). The following additional properties are recognized:

TITLE The title to put on the window. If NIL, the window will not be given a title.

FONT The font in which to paint items (a font descriptor or any other argument acceptable to FONTCREATE). This font is made the window's current font. If the browser's print function displays items in more than one font, the application should choose the tallest font for the FONT property, and it is responsible for ensuring that the correct font is always in use. If this property is NIL, the default display font is used. The browser fields TBFONT and TBFONTHEIGHT are set from this font. In addition, if the caller did not specify the ITEMHEIGHT property, the fields TBITEMHEIGHT and TBBASELINE are calculated from the font. TableBrowser uses the sizes to know where each line starts.

If the caller specified the ITEMHEIGHT property but no BASELINE, the baseline is taken to be zero. If the caller did not specify ITEMHEIGHT, then the baseline is calculated to be the font's descent, or in the case of multiple lines per item, the descent plus the font height times (#LINESPERITEM–1), so that the behavior described above for TB#LINESPERITEM holds.

TB.MAKE.BROWSER returns a new TABLEBROWSER object describing the browser. The application is free to attach further windows to this one. The TABLEBROWSER object is also stored on the window's TABLEBROWSER property.

(TB.REPLACE.ITEMS *BROWSER NEWITEMS*) [Function]

Completely replaces the items of *BROWSER* with *NEWITEMS*, a list of TABLEITEM records. This is a lot like creating a new browser, except that the window structure is already there.

(TB.SET.FONT *BROWSER FONT*) [Function]

Changes *BROWSER*'s display font to *FONT*, which is of the same form as the FONT property given to TB.MAKE.BROWSER. If FONT is NIL, TB.SET.FONT makes its computations based on the browsers current display-related fields (TBFONT, TB#LINESPERITEM, etc).

TB.SET.FONT clears the window; the application is responsible for ensuring that the window is redisplayed, e.g., by calling REDISPLAYW. TB.SET.FONT does not do the redisplay itself, so as to avoid double redisplay in the case where the application also wants to change the items at the same time (by calling TB.REPLACE.ITEMS).

(TB.FINISH.CLOSE *BROWSER WINDOW CLOSEFLG* —) [Function]

Takes care of closing or shrinking *WINDOW*, occupied by *BROWSER*, after the cleanup performed by the browser's TBCLOSEFN (see above).

*CLOSEFLG* is one of the symbols CLOSE or SHRINK.

(TB.BROWSER.BUSY *BROWSER*) [Function]

Briefly changes the cursor to a large "X", the conventional way TableBrowser indicates that an operation attempted with the mouse cannot be performed because the browser is busy.

## Simple Item Operations

(TB.SELECT.ITEM *BROWSER ITEM*) [Function]

Marks *ITEM* in *BROWSER* selected. Ordinarily, selection occurs by means of the mouse. However, applications may want to programmatically select items in response to a user command. Selected items are indicated by a small triangle in the left margin.

(TB.UNSELECT.ITEM *BROWSER ITEM*) [Function]

Marks *ITEM* in *BROWSER* not selected.

(TB.UNSELECT.ALL.ITEMS *BROWSER*) [Function]

Marks *all* items in *BROWSER* not selected. This is considerably faster than unselecting items one at a time. The typical use for this function is prior to calling TB.SELECT.ITEM, to ensure that the new selection is the browser's *only* selection.

(TB.DELETE.ITEM *BROWSER ITEM*) [Function]

Marks *ITEM* in *BROWSER* for deletion. The display shows a line drawn through the item. It is permissible to delete a deleted item (it is a no-op).

(TB.UNDELETE.ITEM *BROWSER ITEM*) [Function]

Removes the deletion mark from *ITEM* in *BROWSER*.

(TB.INSERT.ITEM *BROWSER NEWITEM BEFOREITEM*) [Function]

Adds *NEWITEM* to *BROWSER*'s set of items, inserting it immediately before the item *BEFOREITEM*, or at the end if *BEFOREITEM* is NIL.

(TB.REMOVE.ITEM *BROWSER ITEM*) [Function]

Removes *ITEM* from *BROWSER*'s set of items. This is the operation that an application's "Expunge" function would typically use, but it need not be in any way correlated with deleted items.

(TB.NORMALIZE.ITEM *BROWSER ITEM*) [Function]

Scrolls *BROWSER*'s window, if necessary, so that *ITEM* is visible.

(TB.CLEAR.LINE *BROWSER ITEM LEFT WIDTH*) [Function]

Clears the contents of *ITEM*'s line in *BROWSER*, starting at the x-position *LEFT* and clearing a region *WIDTH* pixels wide. *LEFT* defaults to zero, *WIDTH* to infinity, so omitting both clears the whole line.

This function is typically used by a browser's print function to clear the line before displaying fresh contents. An application wanting to print with minimal visual disruption may want to use TB.CLEAR.LINE only on those portions of the line not being printed to explicitly, so that repainting a line with only slight changes minimizes the apparent display activity.

(TB.REDISPLAY.ITEMS *BROWSER FIRSTITEM LASTITEM*) [Function]

Ordinarily, TableBrowser takes care of deciding when items need to be redisplayed (e.g., when scrolling, reshaping, inserting, etc.). However, there may be times when circumstances beyond the knowledge of TableBrowser require that an item be redisplayed, e.g., when the contents of the item are changed by the application. In such cases, the application is responsible for telling TableBrowser that repainting is needed.

TB.REDISPLAY.ITEMS explicitly invokes *BROWSER*'s repaint method to redisplay items *FIRSTITEM* through *LASTITEM*, which may be the same item in the case of redisplaying a single item. The item arguments may be given as TABLEITEM objects or as a number. *FIRSTITEM* defaults to the browser's first item and *LASTITEM* defaults to the last one; thus (TB.REDISPLAY.ITEMS *BROWSER*) forces redisplay of the entire browser, and is equivalent to calling REDISPLAYW on the window. Only those items currently visible in the window are actually repainted.

## Operations on Multiple Items

(TB.NUMBER.OF.ITEMS *BROWSER TYPE*) [Function]

Returns the number of items in *BROWSER* of the specified *TYPE*, one of the following symbols:

| | |
|---|---|
| NIL | Returns the total number of items. |
| SELECTED | Returns the number of items currently selected. |
| DELETED | Returns the number of items currently deleted. |

(TB.NTH.ITEM *BROWSER N*) [Function]

Returns the *N*th item in *BROWSER*. *N* is an integer; the first item is numbered 1. Returns NIL if *N* is less than 1 or greater than the number of items in the browser.

Most of the following functions accept a predicate or mapping function. The results of the mapping are unpredictable if the mapping function adds or removes items, so an application wishing to do so should first collect the items of interest and map over that list itself.

(TB.COLLECT.ITEMS *BROWSER PREDFN*) [Function]

Returns a list, in browser order, of all the items in *BROWSER* of the type specified by *PREDFN*. *PREDFN* can be one of the symbols NIL, SELECTED or DELETED, which are interpreted as for TB.NUMBER.OF.ITEMS. Otherwise, *PREDFN* is a predicate function of two arguments, *BROWSER* and an item from the browser; *PREDFN* should return T if the item is to be collected.

(TB.MAP.ITEMS *BROWSER MAPFN NULLFN*) [Function]

Applies the function *MAPFN* successively to each item in *BROWSER*. *MAPFN* should accept two arguments, *BROWSER* and the item.

If the browser is empty, TB.MAP.ITEMS instead calls *NULLFN*, if specified, with the single argument *BROWSER*.

(TB.MAP.SELECTED.ITEMS *BROWSER MAPFN NULLFN*) [Function]

Applies the function *MAPFN* successively to each selected item in *BROWSER*. *MAPFN* should accept two arguments, *BROWSER* and the item.

If no items are currently selected in the browser, TB.MAP.SELECTED.ITEMS instead calls *NULLFN*, if specified, with the single argument *BROWSER*.

A typical application calls TB.MAP.SELECTED.ITEMS in response to a menu selection to carry out the operation on the items you have selected.

(TB.MAP.DELETED.ITEMS *BROWSER MAPFN NULLFN*) [Function]

Applies the function *MAPFN* successively to each deleted item in *BROWSER*. *MAPFN* should accept two arguments, *BROWSER* and the item.

If no items are currently deleted in the browser, TB.MAP.DELETED.ITEMS instead calls *NULLFN*, if specified, with the single argument *BROWSER*.

(TB.FIND.ITEM *BROWSER PREDFN FIRST# LAST# BACKWARDSFLG*)[Function]

Returns the first item in *BROWSER* in the range of items numbered *FIRST#* through *LAST#* that satisfies the predicate *PREDFN* (a function of two arguments), *BROWSER* and the item.

*PREDFN* can also be one of the symbols SELECTED or DELETED to search for selected or deleted items.

*FIRST#* defaults to 1, *LAST#* defaults to the number of items in the browser, so omitting both searches the whole browser.

If *BACKWARDSFLG* is true, the range is searched in reverse order.

## Miscellaneous Access Functions

These provide functional access to some of the fields defined in the Records section.

(TB.ITEM.SELECTED? *BROWSER ITEM*) [Function]

Returns T if *ITEM* in *BROWSER* is selected.

(TB.ITEM.DELETED? *BROWSER ITEM*) [Function]

Returns T if *ITEM* in *BROWSER* is deleted.

(TB.WINDOW *BROWSER*) [Function]

Returns a pointer to the window containing the browser.

(TB.USERDATA *BROWSER NEWVALUE*) [Function]

Returns the value of the TBUSERDATA field of the browser; if *NEWVALUE* is supplied, it is stored as the new value of this field.

TB.LEFT.MARGIN [Constant]

The left margin, in pixels, of the start of each item in the browser. Space to the left of this point is used by the selection marker. This constant is compiled into TableBrowser.

## Limitations

In the current implementation, the items in a browser are maintained as a simple list. This means that some operations that might be expected to take constant time (e.g., returning the nth item) instead take linear time (or worse). TableBrowser currently optimizes its operations for sequential access, so that the most typical operations are not adversely affected. However, note that performance may not be acceptable for very large browsers (on the order of 1000 items) when accessing items in random order, or in particular, searching a browser in reverse order.

# TCP-IP

The Transport Control Protocol - Internet Protocol (TCP-IP) family of networking protocols was developed under the auspices of the Department of Defense to standardize communication mechanisms within Department of Defense networks such as the ARPANET.

The protocols are documented in a collection of working papers known as Requests for Comments (RFCs). Appropriate RFC numbers appear throughout this document as new protocols are introduced.

## Requirements

TCP-IP has both hardware and software requirements.

### Hardware

- Ethernet
- Cooperating host (yours or theirs)
- 110X/118X with an Ethernet controller (usually co-resident on an otherwise inhabited module)
- XCVR interface cable
- XCVR installed on an Ethernet with a logical (direct or internet) connection to the cooperating host.

### Software

You need the files enumerated in the section titled "Interlisp Files." Files loaded by the high-level modules TCPFTP, TCPFTPSRV, TCPCHAT, and TCPTFTP automatically load their dependencies. If you load files from floppy, you must load their dependencies first:

| File | Dependencies |
|---|---|
| TCP | TCPLLIP |
| TCPCHAT | TCP, CHAT |
| TCPCONFIG | None |
| TCPDEBUG | TCP |
| TCPDOMAIN | TCPUDP |
| TCPFTP | TCPNAMES, TCP |
| TCPFTPSRV | TCPFTP |
| TCPHTE | None |
| TCPLLAR | None |
| TCPLLICMP | None |
| TCPLLIP | TCPHTE, TCPLLICMP, TCPLLAR |

| | |
|---|---|
| TCPNAMES | None |
| TCPTFTP | TCPUDP |
| TCPUDP | TCPLLIP |

## User Interface

TCP does not have a user interface module of its own. Its functions and variables are accessible via an Interlisp Executive, and you can direct some of its debugging information to a window.

As a network protocol module, it extends capability to other programs which may have their own window interfaces, for example, Chat and FileBrowser.

## Installation

The first step in installing TCP-IP is to add your workstation to a network supporting TCP-IP and communications with others on the net. The rest of this section contains a step-by-step set of directions for this installation.

After you are on the network, load the required .LCOM modules for the type of service you want. For a full description of these modules, see the section "Interlisp Files."

| Module | Implementation |
|---|---|
| TCPFTP | TCP-based file transfer protocol |
| TCPFTPSRV | TCP-based FTP server |
| TCPCHAT | TELNET protocol for the Chat system. |
| TCPTFTP | TFTP protocol. |

### Obtaining Network Addresses

The first thing you need to do is to get a TCP-IP address assigned to each of your workstations from your network administrator. If your site supports Domains, get the name of your local domain and the addresses of your domain server(s) from your network administrator. You will also need to know the network addresses and operating system of the hosts you want to communicate with and the addresses of any network gateways you have.

Note: The maximum length of the domain and organization fields is 20 characters each.

Be sure to find out whether your net is a true Class A, B or C network and is not broken up into subnets. If it is broken up into subnets, be sure to read the discussion on SUBNETMASKs in "A Primer on IP Networks."

**Warning For Sun Installations:** When running TCP-IP to a Sun from an 11xx, directory enumeration on an unmatched directory path returns a listing for the top-level directory of the logged-in user. The TCPFTP protocol does not support directory creation.

## Creating HOST.TXT File

Create a HOSTS.TXT file containing entries for the TCP-IP hosts needed by the user community and place a copy of the file on either a directory contained in the DIRECTORIES search path of each workstation on the net or the local disk of each Interlisp workstation.

The following is a sample HOSTS.TXT file:

```
; Hosts.txt,
; Internet Hosts Table for Networks 192.20.10.0 and 174.23.0.0
; 12-Dec-86
;
; The format of this file is documented in RFC 810, "DoD Internet
; Host Table Specification", which is available online at SRI-NIC
; as the file
;              [SRI-NIC]<RFC>RFC952.TXT
;
; It may be retrieved via FTP using username ANONYMOUS with
; any password.
;
; or as the file
;                                    [INDIGO]<RFC>RFC952.TXT
; Read access to GV World. Valid GV credentials required.
;
; The format for entries is:
;
; GATEWAY: ADDR, ADDR : NAME : CPUTYPE : OPSYS : PROTOCOLS :
; HOST: ADDR, ALTERNATE-ADDR (if any): HOSTNAME,NICKNAME : CPUTYPE :
;   OPSYS : PROTOCOLS :
;
; Where:
;;  ADDR = internet address in decimal, e.g., 26.0.0.73
;;  CPUTYPE = machine type (Xerox-11xx, VAX-11/780, SUN, etc.)
;;   OPSYS = operating system (UNIX, TOPS20, TENEX, VMS, Interlisp,
etc.)
;;  PROTOCOLS = transport/service (TCP/TELNET, TCP/FTP, etc.)
;;  : (colon) = field delimiter
;;  :: (2 colons, NO space between) = null field
;
```

```
HOST : 192.20.10.1 : Bach : Xerox-1108 : Interlisp : TCP/TELNET,
TCP/FTP :
HOST : 192.20.10.3 : PARC-VAXC : VAX-11/780 : UNIX : TCP/TELNET,
TCP/FTP :
HOST : 192.20.10.15 : Oberon : VAX-11/780 : VMS : TCP/TELNET, TCP/FTP
:
HOST : 192.20.10.71 : Explorer : TI-EXPLORER : TOPS-20 : TCP/TELNET,
TCP/FTP :
HOST : 174.23.77.22 : Sunrise : SUN : UNIX : TCP/TELNET, TCP/FTP :
HOST : 174.23.30.21 : Rutgers : VAX-11/780 : TOPS-20 : TCP/TELNET,
TCP/FTP :
HOST : 174.23.76.21 : Simba : SYMBOLICS : SYMBOLICS-3600 :
TCP/TELNET, TFP/FTP :
GATEWAY : 192.20.10.240, 174.23.77.250 : Hellsgate : VMS : IP/GW :
```

This example shows a host table that indicates that there are four hosts (Bach, PARC-VAXC, Oberon, and Explorer) on net 192.20.10.0, three hosts (Sunrise, Rutgers, and Simba) on net 174.23.0.0 and a gateway (Hellsgate) that connects the two.

In regard to the OPSYS field in the HOSTS.TXT file, it is preferable to use values recognized by the Lisp variable NETWORKOSTYPES. Interlisp is the default value if a host's OSType is not declared.

Note that if any host is accessible via another network protocol (for example, PUP or NS), you may desire to call the host by an unambiguous name when it is accessed via TCP. You can do this by giving it an unambiguous name in the HOSTS.TXT file.

If you ever modify the HOSTS.TXT table after TCP.LCOM has been loaded, use the function (\HTE.READ.FILE *'HOSTTABLE*) to reread the file.

For example,

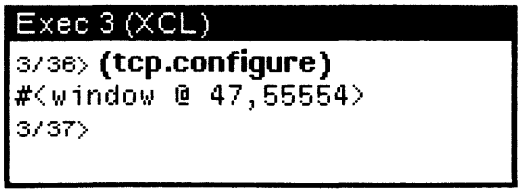
```
(\HTE.READ.FILE '{DSK}<LISPFILES>HOSTS.TXT)
```

TCP.ALWAYS.READ.HOSTS.FILE [Variable]

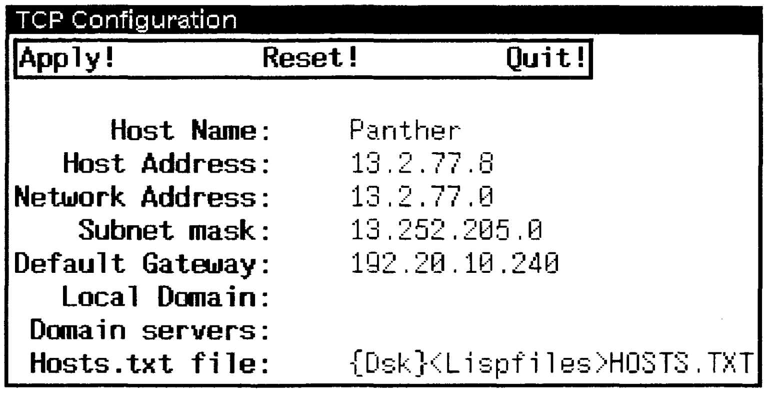
Initially set to T. Setting it to NIL causes the system to parse the HOSTS.TXT file only when the filename (stored in the configuration file) is different from the previously read filename, or the write date of the file has changed. The HOSTS.TXT file will always be read at least once when loading the software into a clean sysout.

## Creating the Local IP.INIT File

TCP.CONFIGURE brings up a menu that you complete.

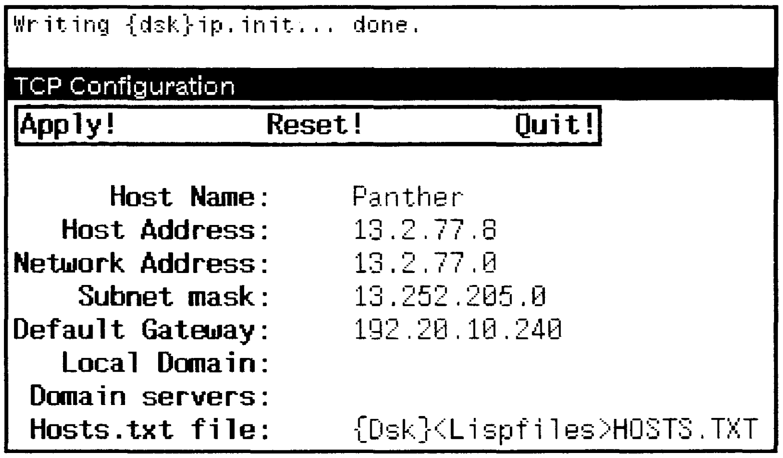


```
Exec 3 (XCL)
3/36> (tcp.configure)
#<window @ 47,55554>
3/37>
```



```
TCP Configuration
Apply!          Reset!          Quit!

      Host Name:     Panther
   Host Address:     13.2.77.8
Network Address:     13.2.77.0
    Subnet mask:     13.252.205.0
Default Gateway:     192.20.10.240
   Local Domain:
 Domain servers:
 Hosts.txt file:     {Dsk}<Lispfiles>HOSTS.TXT
```



```
Writing {dsk}ip.init... done.

TCP Configuration
Apply!          Reset!          Quit!

      Host Name:     Panther
   Host Address:     13.2.77.8
Network Address:     13.2.77.0
    Subnet mask:     13.252.205.0
Default Gateway:     192.20.10.240
   Local Domain:
 Domain servers:
 Hosts.txt file:     {Dsk}<Lispfiles>HOSTS.TXT
```

If any field does not apply to your site, leave it blank.

Selecting **Apply!** writes the file {DSK}<LISPFILES>IP.INIT to the local disk.

Note: The file {DSK}<LISPFILES>IP.INIT must exist on each Interlisp machine before TCP.LCOM is loaded. And this file *must* remain on the workstation and must not be copied to other workstations. Also, the font GACHA 12 MRR must be available.

Selecting **Reset!** resets the menu to the original state.

Selecting **Quit!** closes the window.

You must perform the TCP.CONFIGURE step individually on each workstation, but you need to perform it only once. As long as there is an IP.INIT file on the workstation, the TCP-IP module will be configured automatically whenever it is loaded or initialized.

If you change your IP.INIT file while TCP-IP is running, you will be prompted to confirm **Restarting TCP**. In most cases, you should confirm the restart.

## Adding Host and Operating System Names to NETWORKOSTYPES

The variable NETWORKOSTYPES is used during Chat to determine the sequence of characters to send when performing auto-login. There should be an entry in NETWORKOSTYPES for each TCP host that you want to communicate with in the form (TCPHOSTNAME . OSTYPE).

For example:

```
((SUNRISE . UNIX)(RUTGERS . TOPS-20) etc)
```

## Loading TCP

Make sure the variables DIRECTORIES and LISPUSERSDIRECTORIES point to the location of the .LCOM files of TCP-IP, or that they are in the connected directory.

You can then load TCP.LCOM which in turn loads its dependent files.

If you plan to do TCP file transfers, load TCPFTP.

If you plan to use the 11xx Lisp workstation as a TCPFTP Server host, load TCPFTPSRV. To start the server, evaluate (TCPFTP.SERVER). An Interlisp machine running the TCPFTP server should be identified as a TOPS-20 machine in the other Interlisp machines' HOSTS.TXT table. It will thus masquesrade as a TOPS-20 server.

The rest is automatic. You can treat an Interlisp host running the server just like any other TCPFTP server. The default path for resolving filenames is {DSK}<LISPFILES>, but you can change or override it.

For example, assume {ERIC} is a machine running the FTP server. From another machine which has TCPFTP loaded, you can do SEE {ERIC}{FLOPPY}FOO, which will type out the file FOO located on the floppy drive of {ERIC}.

If you plan to Chat to a TCP host, load CHAT, CHATTERMINAL, DMCHAT and then TCPCHAT.LCOM. Be sure that hosts with which you wish to chat have their NETWORKOSTYPES set.

If you plan to use the TCP Trivial File Transfer Protocol, load TCPTFTP.

Interlisp's TCPTFTP also provides a TCPTFTP server. Load TCPTFTP.LCOM and evaluate (TFTP.SERVER) . You can then use the appropriate TFTP commands to copy files from the Interlisp machine; for example TFTP.PUT and TFTP.GET.

## Verifying TCP Connections

Load TCPDEBUG. Execute (TCPTRACE T) and you will be prompted to open a window to show TCP packets. Select INCOMING, OUTGOING and CONTENTS from the window's menu. If the host that you are communicating with has a TCP echoserver process you can then try (TCP.ECHOTEST 'HOSTNAME

3) . For example, using the above HOSTS.TXT file this would be (TCP.ECHOTEST 'SIMBA 3).

You will be prompted to open a window for the echo test and should see text, for example:

**This is byte number 21**
**This is byte number 45**
**This is byte number 69**

You should also see packets being sent and received in the TCPTRACE window.

Note: If a remote host is not running a TCP echo server process you will not get this response.

## Connecting, Transferring Files, and Chatting to a Host

First log in to the host by typing (LOGIN 'HOST); for example, (LOGIN 'SUNRISE). You will then be prompted for a user name and password. This will be sent to the host when you attempt to CONNect or Chat.

Use the command CONN {HOST}<DIRECTORY>SUBDIR> to connect to a particular host. The local directory delimiters < and > can be used when connecting or file transferring. When communicating with a remote host you can specify the directory path as <DIRECTORY> SUBDIRECTORY> SUBDIRECTORY... , and the appropriate delimiters are presented to the remote host. Determination of what delimiter is presented depends upon the value of the OSTYPE field in the HOSTS.TXT file. If the field is empty, OSTYPE = 'Interlisp' is the default.

Using the above HOST.TXT file as an example you can do the following:

| | |
|---|---|
| UNIX | `CONN {SUNRISE}<DIR>SUBDIR>SUBDIR>` |
| VMS | `CONN {OBERON}<DIR>SUBDIR>SUBDIR>` |
| TOPS-20 | `CONN {RUTGERS}<DIR>SUBDIR>` |
| SYMBOLICS-3600 | `CONN {SIMBA}<DIR>SUBDIR>` |

You can then do a DIR of the remote host, COPYFILE files to and from the host, assuming TCPFTP is loaded, MAKEFILE, etc.

Since the TCPFTP specification does not specify file type conventions, the variable TCP.DEFAULT.FILETYPES is used to associate a file's extension with the type of file it is. It is a list in the form (extension . type); for example,

```
((LCOM . BINARY)  (TXT . TEXT) etc)
```

Since Unix systems are case-sensitive, you should also have the lower case version of the file extensions on this list. If a file extension is not found on this list, the variable TCP.DEFAULTFILETYPE is used as the default file type during file transfers.

To Chat to a remote host, select Chat from the background menu and enter the host name when prompted. You will be prompted for a Chat window and should then be able to chat to the host. If you have problems opening the Chat connection, try (CHAT 'HOST 'NONE). This will suppress the automatic login.

### Making a Sysout that Contains TCP-IP

1. Load Medley sysout.
2. Create TCP host table.
3. Load TCPCONFIG.LCOM and run TCP.CONFIGURE if there is no IP.INIT file on local disk.
4. Load TCP.LCOM.TCPFTP.LCOM, TCPCHAT.LCOM.
5. Load TCPDEBUG.LCOM if you always want to have trace and echo facilities available.
6. Evaluate (TCP.STOP)
7. Evaluate (STOPIP)
8. Evaluate

   (SETQ RESTARTETHERFNS (LIST '(LAMBDA NIL (AND \IPFLG (\IPINIT)))))
9. Load any other files that you want in this sysout.
10. Evaluate SYSOUT to the device of your choice. Evaluate (\TCP.INIT) to re-enable TCP.
11. Load TCP sysout on other machine.
12. Create TCP host table.
13. Evaluate (TCP.CONFIGURE) and identify the new machine.
14. Evaluate (\TCP.INIT) to re-enable TCP.
15. Evaluate (\IPINIT) to restart the IPLISTENER process.

## TCP-IP Protocol Layers

The TCP-IP family consists of four principal protocol layers: the link layer, the internet layer, the transport layer, and the application layer.

### Link Layer

The physical link layer, the medium for transferring packets between hosts, is assumed to be any medium capable of transporting packets of data between hosts. Common link layers in this family include the Ethernet and the ARPANET.

The Address Resolution (AR) Protocol enables hosts to map between internet addresses and link layer addresses.

For example, the internet layer protocol IP (see below) uses a 32-bit combined unique host and network address; the host address field is of variable size and depends on the pattern encoded in the high-order bits of the address. On the other hand, the 10 MB Ethernet uses a fixed-size 48-bit unique host address. The Address Resolution protocol, documented in RFC826, allows hosts to discover dynamically the link layer address equivalents of other internet hosts.

## Internet Layer

The internet layer is responsible for routing packets between hosts. Unlike the link layer, the internet layer is capable of moving packets between hosts that are not connected to the same network. The term IP in TCP-IP refers to the Internet Protocol, the protocol that performs this task in the TCP-IP family. IP is documented in RFC791. IP is not assumed to be error-free; packets may be lost or duplicated while moving through the internet. It is the responsibility of the transport layer (see below) to guarantee perfect delivery, should the client require it.

IP also depends on an associated protocol called the Internet Control Message Protocol (ICMP). ICMP is responsible for handling exception conditions that arise between hosts using IP. Such conditions include the inability to deliver packets, errors in packet formats, etc. ICMP is documented in RFC792.

## Transport Layer

The transport layer is responsible for assuring error-free, duplicate-free, sequenced delivery of packets between communicating processes. The most common transport layer is TCP, the Transport Control Protocol. TCP maintains the appearance of a perfect byte stream between processes. TCP is documented in RFC793.

An unreliable transport layer called the User Datagram Protocol (UDP) allows for packet exchange between communicating processes, but makes no attempt to guarantee delivery, suppress duplication, etc. Clients of UDP must provide their own error-recovery mechanisms if necessary. UDP is documented in RFC768.

## Application Layer

Many applications exist in the TCP-IP family. The most common applications are file transfer, virtual terminal interaction, and mail delivery.

### File Transfer

Two principal file transfer applications are in use: FTP, based on TCP and documented in RFC765; and TFTP (the Trivial File Transfer Protocol), based on UDP and documented in RFC783. Both are implemented in Interlisp, and are discussed at greater length below.

### Virtual Terminal Interaction

The TELNET protocol, documented in RFC854, specifies the protocol for virtual terminal interaction between a user and a remote system. The Chat module will use the TELNET protocol to connect to TCP-only hosts.

### Mail Delivery

The Simple Mail Transfer Protocol (SMTP) enables the delivery of mail between system elements using TCP. It is not currently implemented in Interlisp. SMTP is documented in RFC821. The format of messages is described in RFC822.

## A Primer on IP Networks

The Internet Protocol internetwork is a collection of IP networks, a subset of which may communicate with each other. Each network is assigned an IP address, which is composed of a network number and a host number. No two hosts in the internetwork have the same network and host number combination; the composition of the network and host number for a particular host unambiguously identifies that host within the internetwork.

### Network Addresses

The address space of the internetwork is formed of the concatenated network and host numbers of its constituent hosts, and is 32 bits long. This 32-bit address space is currently partitioned into three classes of network addresses, known as class-A, class-B, and class-C:

Class-A addresses consist of 7 bits of network number and 24 bits of host number.

Class-B addresses consist of 14 bits of network number and 16 bits of host number.

Class-C addresses consist of 21 bits of network number and 8 bits of host number.

Thus, there may be 128 class-A networks, 16,384 class-B networks, and over two million class-C networks. In addition, a single class-A network has the capacity to address over 16 million hosts, while a class-C network can address only 255 hosts. The class to which a particular IP network belongs may be determined by examining the most significant bits of its address.

Network number assignments are strictly controlled by a central authority. Institutions requesting network assignments are given class-A, -B, or -C networks depending on their estimated eventual size (numbers of hosts). Sites without assigned network numbers may request an assigned number by contacting:

Joyce Reynolds
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90292-6695
Phone: (213) 822-1511
ARPANET: JKREYNOLDS@USC-ISIF.ARPA

IP addresses are normally stored or exchanged as single 32-bit numbers. The printed representation of an IP address takes the form W.X.Y.Z, where W through Z are the decimal equivalents of each of the 8--bit bytes that constitute the address. Class-A addresses are of the form N.H.H.H; class-B addresses are of the form N.N.H.H; and class-C addresses are of the form N.N.N.H, where N indicates a byte of the network number, and H indicates a byte of the host number.

Class-A: N.H.H.H The first number is between 0-127 (for example, 122.0.2.1)

Class-B: N.N.H.H The first number is between 128-191 (for example, 153.4.23.5)

Class-C: N.N.N.H The first number is between 192-255 (for example, 194.5.67.3)

For example, 36.47.0.12 is an address on network 36, a class-A network; and 192.10.200.1 is an address on network 192.10.200, a class-C address.

## Broadcast Address

The Internet Protocol defines an address in which the host field contains all ones to be a broadcast address for its network. Thus, the address 36.255.255.255 is the broadcast address on network 36, and 192.10.200.255 is the broadcast address on network 192.10.200.

## Subnets

It is quite common for class-B networks to be partitioned into a set of smaller subnetworks, which are really class-C networks, but have the wrong network number to be recognized as class-C networks. This is just as common is partitioning a class-A network into many class-B subnetworks. An implementation of TCP-IP that is not prepared to handle this violation of the IP standard will not be able to communicate with hosts on the same network but different subnetworks. Fortunately, extending an IP implementation to support subnetworks is straightforward.

SUBNETMASK is a 32-bit parameter that resembles an IP address. The purpose of the mask is to enable a host to determine when a destination IP address is or is not on the same subnet as the sending host itself.

The SUBNETMASK has the following properties:

- The bitwise-AND of a source host's address (for example, this machine) and the SUBNETMASK must be equal to the bitwise-AND of a destination host's address and the

SUBNETMASK if and only if the two hosts are on the same subnetwork.

- The bitwise-AND of a source host's address and the SUBNETMASK must not be equal to the bitwise-AND of a destination host's address and the SUBNETMASK if and only if the two hosts are on different subnetworks.

As an example, consider network 39.0.0.0. This is a class-A network. Suppose this network consists of a number of subnetworks; for example, subnetworks with numbers like 39.47.*.* and 39.9.*.*. According to the IP specification, these subnetworks should really be one monolithic network, such that a host desiring to communicate with any other host whose address begins with 39... should have to take no special action with regard to routing packets to that host. Let us assume that this is not the case. The only way a machine has of telling which hosts are on different networks is to compare the masked version of the address with the masked version of its own address.

To continue the example further, assume the following:

Host A has address 39.9.0.6. Host A's SUBNETMASK is 39.255.0.0.

Host B has address 39.9.0.7. Host B's SUBNETMASK is also 39.255.0.0.

Host C has address 39.47.0.6. Host C's SUBNETMASK is also 39.255.0.0.

When host A sends to host B, it compares its masked address with host B's masked address, and finds them equal:

39.9.0.6 AND 39.255.0.0 = 39.9.0.0; 39.9.0.7 AND 39.255.0.0 = 39.9.0.0

However, when host A sends to host C, it finds the masked comparison does not match:

39.9.0.6 AND 39.255.0.0 = 39.9.0.0; 39.47.0.6 AND 39.255.0.0 = 36.47.0.0

Class-A networks that are subdivided into class-B subnetworks have SUBNETMASKs that look like X.255.0.0, where X is the class-A network number. Likewise, class-B networks subdivided into class-C subnetworks have SUBNETMASKs that look like X.Y.255.0, where X.Y is the class-B network number. Finally, networks in which subnet routing is not in use have SUBNETMASKs identical to their network addresses. For example, if network 36 did not use subnet routing, its SUBNETMASK would be 36.0.0.0.

The definitive document on this approach to subnetwork routing is RFC940.

## Interlisp Files

The files that implement the TCP-IP protocol suite are divided into two classes: those that implement low-level functionality, normally not of interest to general users, and those that implement higher-level functionality for user programs (either application or transport layer protocols).

The higher-level functions reside in the files TCP, TCPDEBUG, TCPFTP, TCPCHAT, TCPNAMES, TCPPUDP, and TCPFTP.

| | |
|---|---|
| TCP | The TCP layer. Implements TCP streams, based on the buffered TCP device (for example , BIN runs in microcode). |
| TCPDEBUG | Contains routines to help debug TCP and TCP-based applications. |
| TCPFTP | Contains the TCP-based file transfer protocol. Creates a new virtual I/O device, allowing transparent filing operations with TCP-only hosts. |
| TCPFTPSRV | Contains the TCP-based FTP server program. When the server program is running on a Xerox 1100-series workstation, other TCP-based hosts may transfer files to and from the workstation. |
| TCPNAMES | Implements translation of file name formats between operating system types. |
| TCPCHAT | Implements the TELNET protocol for the Chat system. |
| TCPUDP | Contains the UDP layer. |
| TCPTFTP | Implements the TFTP protocol. Creates a buffered TFTP device to allow efficient bulk transfer between hosts. |

The low-level functions reside in the files TCPLLIP, TCPLLICMP, TCPLLAR, TCPHTE, and TCPCONFIG.

| | |
|---|---|
| TCPLLIP | Implements the IP layer. |
| TCPLLICMP | Implements ICMP for IP. |
| TCPLLAR | Implements AR for the 3- and 10-megabyte Ethernets. |
| TCPHTE | Implements the functionality necessary to parse RFC810-style HOSTS.TXT files. This allows name-to-address translation within the Interlisp host. |
| TCPCONFIG | Provides a function to carry on a configuration dialog when TCP-IP is first installed on a machine. This file needs to be loaded only once, to produce the file {DSK}IP.INIT. Thereafter, TCPCONFIG is needed only to reestablish or modify IP parameters. |

## TCP

TCP implements the transport control protocol for Interlisp. After TCP is loaded, Interlisp supports a TCP stream capable of bidirectional I/O to a remote system element. The following functions are intended for use by applications programs.

(TCP.OPEN *DST.HOST DST.PORT SRC.PORT MODE ACCESS NOERRORFLG OPTIONS*) [Function]

Opens a TCP stream to *DST.PORT* on *DST.HOST* from *SRC.PORT*.

*DST.HOST* can be a host name, an IP host address in text format (such as 192.10.200.1), or the 32-bit integer representation of an IP host address as returned by the function DODIP.HOSTP (which is documented under TCPLLIP).

*DST.PORT* is a 16-bit number representing a TCP port open in LISTENING mode on the remote system.

*SRC.PORT* is also a 16-bit number, but may be supplied as NIL to obtain a defaulted unique local port number.

*MODE* is either ACTIVE, meaning to act as initiator of the connection, or PASSIVE, meaning to wait for a remote system element to initiate the connection.

*ACCESS* is either INPUT, OUTPUT, or APPEND (OUTPUT and APPEND are treated in the same manner).

If *NOERRORFLG* is non-NIL, TCP.OPEN will return NIL if the connection fails; otherwise, TCP.OPEN will call ERROR to signal failure.

*OPTIONS* is an optional parameter which allows the application program to control some of the characteristics of the TCP connection. *OPTIONS* is supplied in property-list format. Currently, the only recognized option is MAXSEG, whose value should be the number of data bytes the remote TCP sender is allowed to place into a single TCP segment (Ethernet packet). The maximum value of MAXSEG is 536.

If TCP.OPEN succeeds, it returns a STREAM open as specified by *ACCESS*. The generic operations BIN, BOUT, PEEKBIN, BINS, BOUTS, READP, EOFP, OPENP, GETFILEPTR, FORCEOUTPUT, and CLOSEF may be performed on streams opened for suitable access.

(TCP.OTHER.STREAM *STREAM*) [Function]

Returns the STREAM open in the other direction with respect to *STREAM* (for example, if *STREAM* is open for INPUT, TCP.OTHER.STREAM returns a STREAM open for OUTPUT, and vice versa).

(TCP.URGENT.EVENT *STREAM*) [Function]

Returns an event upon which a user process may wait for URGENT data to arrive on *STREAM*.

(TCP.URGENTP *STREAM*) [Function]

Returns T if *STREAM* is currently reading URGENT data.

(TCP.URGENT.MARK *STREAM*) [Function]

Marks the current point in *STREAM* as the end of URGENT data. *STREAM* must be open for OUTPUT.

(TCP.CLOSE.SENDER *STREAM*) [Function]

Closes the output side of *STREAM*, which may be either the INPUT or OUTPUT stream for the connection. This function differs from CLOSEF in that the INPUT side of the connection is not closed (although the remote system element may close the connection once the local output side of the connection is closed).

(TCP.STOP) [Function]

Disables the TCP protocol, closing all open TCP streams.

(\TCP.INIT) [Function]

(Re)initializes the TCP module.

\TCP.DEFAULT.RECEIVE.WINDOW [Variable]

Is the default number of bytes allowed outstanding from the remote system. It is initially 4,096.

\TCP.DEFAULT.USER.TIMEOUT [Variable]

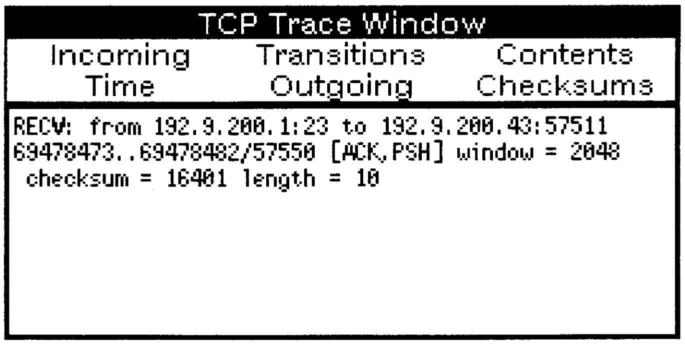
Is the default number of milliseconds a remote system element is allowed to remain silent before the TCP connection is declared broken. It is initially 60,000.

## TCPDEBUG

TCPDEBUG implements tracing and test functions used to debug TCP and TCP-based applications.

(TCPTRACE) [Function]

Opens a trace window and attaches a menu to the window's top.



```
RECV: from 192.9.200.1:23 to 192.9.200.43:57511
69478473..69478482/57550 [ACK,PSH] window = 2048
 checksum = 16401 length = 10
```

The menu entries represent state changes or data elements to be traced; each entry is a toggle. Clicking on the toggle once will activate the trace of the particular element and will gray-over the entry; clicking a second time will deactivate the tracing and ungray the menu item. The following data elements/transitions may be displayed:

| | |
|---|---|
| Contents | Displays a line's worth of packet contents. The Incoming or Outgoing switch must be on. |
| Incoming | Displays incoming data. |
| Outgoing | Displays outgoing data. |

| | |
|---|---|
| Checksums | Displays checksums for each TCP segment. |
| Time | Displays the time interval since the last action on the connection. |
| Transitions | Displays state transitions on the TCP state machine. |

(PPTCB *TCB FILE*) [Function]

Prints the state of a TCP connection. PPTCB is normally the INFO function for the process that monitors a connection; thus, selecting INFO in the process status window will cause a window to pop up containing a report on the status of the associated connection.

(TCP.ECHOTEST *HOST NLINES*) [Function]

Opens a TCP connection to the TCP echo port on *HOST* and sends *NLINES* of random text. The echo responses are displayed in a window. If *NLINES* is NIL, the echo test will run forever.

(TCP.ECHO.SERVER *PORT*) [Function]

Starts a TCP echo server on *PORT* (defaults to the TCP echo port). It is usually more useful to start the echo server as a process by doing (ADD.PROCESS '(TCP.ECHO.SERVER *PORT*)).

(TCP.SINK.SERVER *PORT*) [Function]

Starts a TCP sink server on *PORT* (defaults to the TCP sink port). Any data sent to this port will be acknowledged and discarded. As with the TCP echo server, it is usually more useful to start this server as an independent process.

(TCP.FAUCET *HOST PORT NLINES*) [Function]

If *HOST* is non-NIL, this function opens a connection to *PORT* on *HOST* and sends *NLINES* of text (the default is to send lines of text forever). *PORT* defaults to the TCP sink port. If *HOST* is NIL, this function waits for a remote system to connect to the TCP faucet port and then sends out *NLINES* of random text.

## TCPFTP

TCPFTP implements a virtual I/O device that performs Lisp filing operations transparently using the RFC765 FTP protocol. The standard filing operations of reading, writing, renaming, deleting, and directory enumeration are supported by the TCPFTP device. However, neither random access filing nor GETFILEINFO are supported, as there is no protocol specification for performing these operations on files. Interlisp operations such as RECOMPILE will not work when files are stored on TCPFTP file servers.

Once TCPFTP is loaded, filing operations should be transparent to users; no additional initialization need be performed. There are, however, two important global variables:

TCPFTP.DEFAULT.FILETYPES [Variable]

This variable is an association list, keyed by common extensions of file names, and contains appropriate file types (for example, TEXT or BINARY) for such files. The TCPFTP protocol provides no mechanism for determining the type of a file about to be retrieved. The file type is usually known in the case of output operations (for example, COPYFILE or MAKEFILE to a file server). However, in the case of COPYFILE from a file server, the TCPFTP module has to infer the file type from other knowledge. The module tries to match the extension of the file name with an entry on the list TCPFTP.DEFAULT.FILETYPES. If it finds a match, it uses the value of the entry in the list as the file type of the file; if it doesn't find a match, it uses the value of TCP.DEFAULTFILETYPE for the file type of the file.

TCP.DEFAULTFILETYPE [Variable]

If no matching extension is found for the file being opened, the TCPFTP module uses the value of TCP.DEFAULTFILETYPE as the file type of the remote file. The initial value of TCP.DEFAULTFILETYPE is BINARY; however, users may preset its value in their INIT.LISP files prior to loading TCP-IP.

The following functions are available for debugging broken file server connections.

(FTPDEBUG *FLG*) [Function]

If *FLG* is T, this function opens a scrolling trace window that displays FTP commands as they are issued. PUPFTP commands will also be displayed in this window (the window is the value of FTPDEBUGLOG).

(\TCP.BYE *HOST*) [Function]

Breaks an FTP connection to *HOST*.

(\TCPFTP.INIT) [Function]

(Re)initializes the TCPFTP module.

## TCPFTPSRV

The TCPFTPSRV module contains a program which implements an FTP service for Interlisp. When this program is running on a workstation, other hosts are able to store and retrieve files from the workstation.

(TCPFTP.SERVER *PORT DEFAULT.FILE.PATH*) [Function]

To start the server program, evaluate the form (TCPFTP.SERVER). If *PORT* is supplied, the FTP server program will listen for connections on the TCP port specified by *PORT*; otherwise, the server will listen on the default FTP server port, port 21.

If *DEFAULT.FILE.PATH* is supplied, the initial path for resolving file names will be relative to *DEFAULT.FILE.PATH*; the default value of this variable is {DSK}<LISPFILES>.

TCPFTP.SERVER.USE.TOPS20.SYNTAX [Variable]

This variable controls whether file names sent back to FTP client programs are formatted in Tops-20 or Interlisp syntax. If the variable is true (the default), all file names will be formatted in Tops-20 syntax. This permits an Interlisp workstation to masquerade as a Tops-20 mainframe for the purposes of file transfer to and from other vendors' machines.

## TCPNAMES

The TCPNAMES module provides a set of functions for translating between the file-naming conventions of different operating systems. This is needed by the TCPFTP module in order for it to convert between Interlisp format file names and the file name formats of other operating systems.

(REPACKFILENAME.STRING *NAME FOROSTYPE*) [Function]

*NAME* is a file name in some operating system's format. *FOROSTYPE* is the name of an operating system. REPACKFILENAME.STRING attempts to translate *NAME* into a format acceptable to the operating system named by *FOROSTYPE*. *NAME* may be a string or atom; the function always returns a string.

Currently acceptable operating system types are:

IFS
INTERLISP
MS-DOS
SYMBOLICS-3600
TENEX
TOPS-20 (also TOPS20)
UNIX
VMS

(TI-Explorers should use TOPS-20 as their operating system.)

The correspondence between the target operating system type and the file name translation function is maintained in an extensible hash table.

(\REPACKFILENAME.NEW.TRANSLATION *OSTYPE FUNCTION*) [Function]

This function adds a new file name translation function for a new operating system type. The function must be a LAMBDA-NOSPREAD function, and must be prepared to receive either a single property-list format argument, such as would be returned by UNPACKFILENAME, or an arbitrary number of arguments in property-list format.

File names in the above format will be passed to the translation function adhering to the conventions of many operating systems; the function must recognize the operating system type and produce the desired output format, which must be a string.

\REPACKFILENAME.OSTYPE.TABLE [Variable]

This variable is the hash table that stores the correspondence between operating system types and translation functions.

## TCPCHAT

TCPCHAT implements the TELNET protocol for virtual terminal I/O between Interlisp and a remote system. Once loaded into

Interlisp, the standard Chat system will use TCP TELNET to communicate with hosts that are believed to support the protocol.

No user-callable functions reside in this module, although the following variables may be of interest.

TCPCHAT.TELNET.TTY.TYPES [Variable]

This variable is an association list that maps internal names of Chat terminal emulators to official terminal names as specified in RFC884, the TELNET Terminal Type Option. This allows TCPCHAT to set the user's terminal type automatically when a connection is established.

TCPCHAT.TRACEFLG [Variable]

If this variable is non-NIL, TELNET negotiations will be printed to TCPCHAT.TRACEFILE (see below). This is sometimes useful in debugging negotiation problems.

TCPCHAT.TRACEFILE [Variable]

TELNET negotiations are printed to this file if TCPCHAT.TRACEFLG is non-NIL.

## TCPUDP

UDP implements the user datagram protocol. The following functions are meant to be called by client applications.

(UDP.INIT) [Function]

Initializes the UDP module. This function is normally called when UDP is loaded and should not need to be called again under normal circumstances.

(UDP.STOP) [Function]

Disables the UDP module, closing any open UDP sockets.

(UDP.OPEN.SOCKET *SKT# IFCLASH*) [Function]

Opens a socket for UDP operations.

*SKT#*, if supplied, is a 16-bit number and will default to a number between 1,000 and 65,535.

*IFCLASH* specifies what to do if the requested socket is already open and is handled as in OPENPUPSOCKET and OPENNSOCKET (see the *IRM*).

It returns an instance of an IPSOCKET.

(UDP.CLOSE.SOCKET *IPSOCKET NOERRORFLG*) [Function]

Closes an open *IPSOCKET*. If *IPSOCKET* is not an open socket and *NOERRORFLG* is NIL, an error will occur; otherwise, NIL is returned if the socket is not active, and T is returned if the socket is active.

Any remaining packets on the socket's input queue are discarded when this function is called.

(UDP.SOCKET.EVENT *IPSOCKET*) [Function]

Returns an event that a process may use to wait for packet arrival on *IPSOCKET*.

(UDP.SOCKET.NUMBER *IPSOCKET*) [Function]

Returns the socket number of *IPSOCKET*.

(UDP.GET *IPSOCKET WAIT*) [Function]

Returns the next packet waiting on *IPSOCKET*. If no packets are waiting, does one of the following based on the value of *WAIT*.

| | |
|---|---|
| NIL | Returns immediately. |
| T | Waits forever for a packet to arrive. |
| A | *FIXP* waits up to *WAIT* milliseconds for a packet to arrive and returns NIL if none arrived during that time. |

Thus, this function is like GETPUP and GETXIP.

(UDP.SETUP *UDP DESTHOST DESTSOCKET ID IPSOCKET REQUEUE*) [Function]

Initializes a fresh packet (as returned from \ALLOCATE.ETHERPACKET). The packet will be sent to *DESTSOCKET* on *DESTHOST*.

*ID* is a number to be placed in the IP header ID field (zero is fine).

*REQUEUE* specifies what to do with the packet after it is sent; NIL (the default) means no special treatment; FREE means to release the packet and return it to the free packet queue. Any instance of a SYSQUEUE will cause the packet to be queued on the tail of the specified queue.

UDP.SETUP initializes all IP and UDP fields and sets the packet up as a minimum-length UDP packet.

(UDP.SEND *IPSOCKET UDP*) [Function]

Sends *UDP*, a UDP-formatted packet, out from *IPSOCKET*.

(UDP.EXCHANGE *IPSOCKET OUTUDP TIMEOUT*) [Function]

Sends *OUTUDP* out from *IPSOCKET* and waits *TIMEOUT* milliseconds for a response; returns NIL if no response came in during the specified interval, or the packet that did come in during that time.

Clears the socket's input packet queue before waiting for a packet to arrive.

(UDP.APPEND.BYTE *UDP BYTE*) [Function]

Appends *BYTE* to the UDP data portion of *UDP* and increments the UDP and IP length fields by one.

(UDP.APPEND.WORD *UDP WORD*) [Function]

Appends *WORD* to the UDP data portion of *UDP* and increments the UDP and IP length fields by two.

(UDP.APPEND.CELL *UDP CELL*) [Function]

Appends *CELL* to the UDP data portion of *UDP* and increments the UDP and IP length fields by four.

(UDP.APPEND.STRING *UDP STRING*) [Function]

Appends *STRING* to the UDP data portion of *UDP* and increments the UDP and IP length fields by the length *STRING*.

## TCPTFTP

TFTP implements the trivial file transfer protocol. This protocol is useful for transferring unimportant files rapidly (for example, between workstations and printers). The following user-callable functions exist.

(TFTP.PUT *FROM TO PARAMETERS*) [Function]

Sends a file to a TFTP host.

*FROM* may refer to any accessible file; *TO* must refer to a file accessible via TFTP.

No attempt is currently made to translate between Interlisp file name syntax and remote system file name syntax for *TO*.

For example, if *TO* resides on a Unix host, it would take a syntax like {HOST}/DIRECTORY/SUBDIRECTORY/FILENAME.

PARAMETERS is currently a list of parameters in the same format used by OPENFILE in .PARAMETERS; for example ((EOLCONVENTION 1) (TYPE TEXT)).

Note: TFTP transfers between Xerox Lisp and Unix hosts initiated from Xerox Lisp should have the PARAMETERS argument be '((EOLCONVENTION 10)).

(TFTP.GET *FROM TO PARAMETERS*) [Function]

Gets a file from a TFTP host. *FROM* must be a file accessible by TFTP; *TO* may be any file.

The file name syntax caveats for *FROM* are the same as for *TO* in TFTP.PUT. *PARAMETERS* is also as in TFTP.PUT.

(TFTP.SERVER *LOGSTREAM*) [Function]

Starts a TFTP server process.

*LOGSTREAM* may be left NIL, causing a new window to appear when the TFTP server is first invoked. Remote systems that support TFTP clients may store or retrieve files through any Interlisp workstation running the TFTP server.

The full Interlisp syntax for file names is supported; thus, requests to store files whose names include hosts will result in the Interlisp workstation's transparently storing the files on the designated hosts.

(\TFTP.OPENFILE *FILENAME ACCESS RECOG PARAMETERS*) [Function]

Returns a STREAM to open for *ACCESS* on *FILENAME*.

*PARAMETERS* is the usual format; TYPE is the only recognized parameter (BINARY opens a stream in *octet* format; TEXT, the default, opens a stream in NETASCII format; see RFC783).

BIN, BOUT, READP, EOFP, etc., may be used on this stream.

The stream is not RANDACCESSP.

(\TFTP.CLOSEFILE *STREAM*) [Function]

Closes the open stream. This is normally useful for streams open for OUTPUT; for INPUT streams, end-of-file will occur eventually.

## TCPLLIP

For users planning implementations on top of IP, the following low-level TCP functions are available.

### IP Socket Access

(\IPINIT) [Function]

Reinitializes the IP world; for example, after some catastrophe.

(STOPIP) [Function]

Disables IP.

(DODIP.HOSTP *NAME*) [Function]

If *NAME* is an integer, *NAME* is returned unaltered. If *NAME* is a text format IP host address (such as 192.10.200.1), DODIP.HOSTP returns its integer representation.

If *NAME* is a string or atom name, DODIP.HOSTP attempts to convert *NAME* to its IP host address integer value, using information supplied in the HOSTS.TXT file (see TCPHTE, below).

If *NAME* is unknown, DODIP.HOSTP returns NIL.

If *NAME* is known, it is cached with its corresponding address so that the function IPHOSTNAME may be used later to convert the address back to a name.

(IPHOSTNAME *IPADDRESS*) [Function]

Tries to convert *IPADDRESS* to a host name.

If *IPADDRESS* has no known name, it is converted to the text representation of an IP address (for example, 192.10.200.1).

(IPTRACE *MODE*) [Function]

Turns on tracing of IP activity. This function is like PUPTRACE and XIPTRACE, which are documented in the *IRM.*

If *MODE* is NIL, IP tracing is disabled.

If *MODE* is T, verbose IP tracing is enabled.

If *MODE* is PEEK, concise IP tracing is enabled. If *MODE* is either T or PEEK, the user is prompted for a window into which trace output will be printed.

(\IP.ADD.PROTOCOL *PROTOCOL SOCKETCOMPAREFN NOSOCKETFN INPUTFN ICMPFN*) [Function]

Defines a new IP-based protocol. The lowest-level IP functions maintain a list of active protocols and perform packet delivery based on the existence of open sockets for protocols of received packet types.

*PROTOCOL* is a protocol number, a number between 1 and 255. The following protocols are defined and should not be disturbed:

| | |
|---|---|
| TCP | 6 |
| ICMP | 1 |
| UDP | 17 |

*SOCKETCOMPAREFN* is a function with two arguments, an IP packet that has just been received and an open IPSOCKET. This function should return NIL if the packet does not belong to the supplied socket, or T if it does. The function will typically be interested in the IPSOCKET field of the IPSOCKET.

*NOSOCKETFN* is a function with one argument, an IP packet that has just been received. Its purpose is to handle received packets for which no socket can be found. If *NOSOCKETFN* is NIL, the default function, \IP.DEFAULT.NOSOCKETFN, will be used; this function simply returns an ICMP message indicating the socket is unreachable.

*INPUTFN* is a function with two arguments, a received IP packet and an open IPSOCKET. The *INPUTFN* is supposed to handle reception of packets when their destination socket has been found. If *INPUTFN* is NIL, the default function, \IP.DEFAULT.INPUTFN, will be supplied.

*INPUTFN* enqueues the received packet on the IPSQUEUE field of the IPSOCKET if the current queue length (stored in the IPSQUEUELENGTH field) is less than the allocated length (stored in the IPSQUEUEALLOC field).

*INPUTFN* also increments the IPSQUEUELENGTH field, and notifies the event stored in the IPSEVENT field.

*ICMPFN* is a function with two arguments and is called when an ICMP packet referring to the protocol is received. The first argument is a pointer to the received ICMP packet. The second argument is a pointer that may be used as if pointed to the original outgoing packet included in the ICMP data. This allows the protocol functions to parse the data in the ICMP packet to determine which socket sent the offending packet. The *ICMPFN* must never attempt to deallocate the packet identified by the second argument; however, it is quite permissible (and expected) that the *ICMPFN* will release the packet identified by the first argument. The default *ICMPFN* simply releases the packet identified by the first argument.

\IP.ADD.PROTOCOL returns an IPSOCKET datum, which represents the active protocol; it is not in fact a useful IPSOCKET and may be safely ignored.

(\IP.DELETE.PROTOCOL *PROTOCOL*) [Function]

Deactivates a protocol with protocol number *PROTOCOL*. Any open sockets are closed.

(\IP.OPEN.SOCKET *PROTOCOL SOCKET NOERRORFLG SOCKETCOMPAREFN NOSOCKETFN INPUTFN*) [Function]

Attempts to open an IPSOCKET for protocol *PROTOCOL*.

*SOCKET* is the identifying information for this socket; this quantity will be EQUAL-compared with other sockets open on *PROTOCOL*. Should a match be found, an error will occur unless *NOERRORFLG* is T, in which case the existing socket will be returned.

*SOCKETCOMPAREFN*, *NOSOCKETFN*, and *INPUTFN* may be supplied to override the functions specified when the protocol was defined; they are not normally useful, however.

(\IP.CLOSE.SOCKET *SOCKET PROTOCOL NOERRORFLG*) [Function]

Closes a socket open on *PROTOCOL*. *SOCKET* is the same quantity passed to \IP.OPEN.SOCKET; it is currently not an instance of an IPSOCKET. If *NOERRORFLG* is T, an error will not occur if the socket is not found.

## IP Packet Building

The following functions are useful for placing bytes into IP packets (as allocated by \ALLOCATE.ETHERPACKET).

Note that most applications will probably want to define a block record to overlay the data portion of an IP packet. Here is an example of such a block record.

Note: Users who are developing new IP-based protocols will need to load EXPORTS.ALL from the library.

```
(ACCESSFNS UDP
    ((UDPBASE (\IPDATABASE DATUM)))
    (BLOCKRECORD UDPBASE
        ((UDPSOURCEPORT WORD)
        (UDPDESTPORT WORD)
        (UDPLENGTH WORD)
        (UDPCHECKSUM WORD)))
    (ACCESSFNS UDP
        ((UDPCONTENTS
        (\ADDBASE
            (\IPDATABASE DATUM)
            (FOLDHI \UDPOVLEN BYTESPERWORD))))))
```

(\IP.APPEND.BYTE *IP BYTE INHEADER*) [Function]

Appends *BYTE* to the IP data portion of *IP* and increments the IP length field by one. If *INHEADER* is T, the IPHEADERLENGTH field is appropriately incremented so that the bytes appear to have been appended to the options portion of the IP header. There must not be any data bytes in the data portion of the packet if this function is to work correctly.

(\IP.APPEND.WORD *IP WORD INHEADER*) [Function]

Appends *WORD* to the IP data portion of *IP* and increments the IP length field by two. *INHEADER* is as in \IP.APPEND.BYTE.

(\IP.APPEND.CELL *IP CELL INHEADER*) [Function]

Appends *CELL* to the IP data portion of *IP* and increments the IP length field by four. *INHEADER* is as in \IP.APPEND.BYTE.

(\IP.APPEND.STRING *IP STRING*) [Function]

Appends *STRING* to the IP data portion of *IP* and increments the IP length field by the length *STRING*.

### IP Packet Sending

(\IP.SETUPIP *IP DESTHOST ID SOCKET REQUEUE*) [Function]

Initializes *IP*. This function should be called just after *IP* is obtained from \ALLOCATE.ETHERPACKET; if this is not done, the append functions above will fail.

*DESTHOST* is the 32-bit IP address to which this packet will be sent.

*ID* is an arbitrary 16-bit quantity that will become the IPID field of the packet.

*SOCKET* is the open IPSOCKET from which the packet will be sent.

*REQUEUE* defaults to FREE and controls the disposition of the packet after transmission (see the *IRM* for the documentation of SETUPPUP or FILLINXIP).

(\IP.TRANSMIT *IP*) [Function]

Tries to send *IP*. Performs IP checksum algorithm prior to sending. Returns NIL if successful, otherwise it returns a status indication, such as NoRouting or AlreadyQueued. This function is like SENDPUP and SENDXIP, except that no socket argument is required.

## TCPHTE

HTE provides functions for parsing HOSTS.TXT files as documented by RFC810. This file is loaded automatically by LLIP and is used by \IPINIT to read in the initial file, HOSTS.TXT. The following variable and function may be of interest.

HOSTS.TEXT.DIRECTORIES [Variable]

Is the search path for the file HOSTS.TXT. This variable is initialized to NIL; thus the search path to be used is by default DIRECTORIES.

(\HTE.READ.FILE *FILE WANTEDTYPES*) [Function]

Reads a HOSTS.TXT file.

*WANTEDTYPES* is a list of types drawn from the set *{HOST, NET, GATEWAY}*, to be read from the file; types not specified in *WANTEDTYPES* are ignored. *WANTEDTYPES* defaults to *(HOST)*.

## TCP Debugging Aids

With TCPDEBUG loaded use (TCPTRACE T) to open up a trace window of TCP traffic. The appropriate items need to be selected from the windows menu in order for data to be seen.

(SETQ TCPCHAT.TRACEFLG T) will print TELNET negotiations to a file, which is what the variable TCPCHAT.TRACEFILE points to.

(FTPDEBUG T) opens a scrolling trace window that displays FTP commands as they are issued. You will see unencrypted passwords if they are issued.

## Limitations

You must use the 1186 microcode in order for TCP-IP to work on an 1186 (microcode for the 1185 will not do).

TCP-IP will not work with Unix systems that have trailer encapsulation enabled. Connections will hang and then eventually break.

Directory enumeration on a VMS system results in NIL.

### Known Problems in TCPFTPSRV

It does not handle error conditions in the middle of file transfers.

Doing a DIR gives you only filename and version: no author, creation date, etc. This is because the TCPFTP protocol specification doesn't support author, creation date, etc.

If there are multiple files on the system, deleting a file without specifing a specific version deletes the most recent version. The workaround is to give the specific version to delete.

The subdirectory structure is not presented back to the client host. If you have a file on both the <lispfiles> directory and a subdirectory, when you do a DIR *.* you do not see the subdirectory listed, but you do see that there are two files on the host with the same version number.

## References

Users with access to the ARPANET may retrieve any RFC from host SRI-NIC.ARPA with the file transfer protocol (FTP) anonymous log-in option. RFCs are stored under <RFC>RFC*nnn*.TXT, where *nnn* is replaced by the number of the particular RFC.

From points on the Xerox internet, the RFC files can be retrieved from {Indigo}<RFC>. {Indigo} is an IFS host. From Lisp, you can simply (LOGIN) and supply your GV credentials if you haven't already, open a FileBrowser on that directory, and retrieve the file to the local workstation environment.

The following RFCs are mentioned in this manual:

RFC765 (superceded by RFC 959)
RFC768
RFC783
RFC791
RFC792
RFC793
RFC810 (superceded by RFC 952)
RFC814
RFC821
RFC822
RFC826
RFC854
RFC894
RFC895
RFC903
RFC904
RFC940

[This page intentionally left blank]

# TELERAID

TeleRaid is an interactive debugger which can be used either to examine, from one workstation, the state of another workstation's virtual memory, or to look inside a sysout file.

## Requirements

REMOTEVMEM, READSYS, RDSYS, VMEM

Either:

Ethernet PUP connection between the two machines. The machine which is to be examined must be in the TeleRaid mode; i.e., the shape of its cursor must be the TeleRaid prompt. It must also have a PUP address.

Or:

A sysout file.

## Installation

Load TELERAID.LCOM and the required .LCOM modules from the library.

## User Interface

The standard use of TeleRaid is to debug a workstation that has stopped at a maintenance panel halt. Pressing the UNDO key when the machine is in this state transfers control to a small TeleRaid server that responds to simple commands over the network. While the TeleRaid server is running, the cursor changes to TELERAID. Also, on a 1108 workstation, the previous contents of the maintenance panel are restored.

On a 1108 workstation, the maintenance panel halt condition is indicated by a four-digit code that begins with a 9. On an 1186, the four-digit code is displayed at the cursor.

The term "debuggee" is used to denote the sysout file or machine running the TeleRaid server, i.e., the one being debugged, while "debugger" refers to the machine that is viewing the debuggee's virtual memory (usually by running TeleRaid).

## Function

(TELERAID *HOST RAIDIX*) [Function]

Enters an interactive debugger viewing the virtual memory of *HOST*, which must denote a machine running a TeleRaid server. *HOST* is either a host name or a PUP address.

*RAIDIX* is an optional number denoting the radix in which values are printed and numbers are accepted as input; if not specified, it defaults to 8 (octal). The only other accepted value for *RAIDIX* at present is 16, for hexadecimal input and output.

If you don't know a machine's PUP name or address, you can find out by typing control-P on the debuggee: control-P changes the maintenance panel to show the machine's PUP host number in decimal radix. You can also find out your PUP address when Lisp is running (rather than in a maintenance panel halt) by evaluating (PORTSTRING (ETHERHOSTNUMBER)). Users typically do this once and tape a note to the terminal so as to have this information handy.

If the debugger is on the same physical Ethernet as the debuggee, you can use that PUP host number directly as the *HOST* argument. Otherwise, you must convert the PUP host number to octal and use the general form of a PUP address, which is a string of the form *"net#host#"*.

For example, (TELERAID 12) debugs the machine whose PUP address is 12 decimal on the same network. (TELERAID "13#14#") debugs host 14 octal (12 decimal) on network 13 octal.

Note: If the control-P command displays zero in the maintenance panel, it means the machine does not have a PUP host number assigned, or the halt occurred so quickly after booting that the Ethernet has not been fully initialized. In this case, TeleRaid cannot be used. See the description of READSYS (below) for directions on TeleRaiding a sysout file.

## TeleRaid Commands

Each TeleRaid command is a single character, followed by arguments appropriate to the command. In the description of the commands that follows, unless otherwise specified, numbers are assumed to be typed in the default radix (octal unless you have specified a different *RAIDIX* in the call to TELERAID).

### Displaying a Stack

For casual users, the L command followed by several F commands generally provide the most useful information. Many of the other commands require some knowledge of the internal representation of Lisp objects and stack frames, something that this document does not attempt to provide.

L shows the stack of the debuggee, as a back trace consisting of a numbered sequence of frame names. The first frame is usually \MP.ERROR if you got here by a maintenance panel halt.

In the case of MP code 9305, the stack shown is the page fault handler's and is uninteresting, except for the argument to the \INVALIDADDR frame.

Use the control-L *P* command to see the stack of the process that took the fault.

control-L *type* Shows the stack of the debuggee starting at some other place. The argument *type* is a single letter denoting which stack to view. The system has a number of special contexts, which are areas of stack space used by certain system routines.

Legal values of *type* are P (page fault), G (garbage collector), K (keyboard handler), H (hard return), S (stack manipulator), R (reset), and M (miscellaneous).

The most interesting of these for most users is P, which for MP code 9305 shows the stack in which the address fault occurred. In addition, *type* F lets you view the stack starting at an arbitrary stack frame; follow F with an octal number denoting the frame (as in the control-X command, below).

K *type* Changes the type of stack link that the L and control-L commands follow to be *type*, which is either A or C. The default is to follow CLinks (control links). ALinks follow the chain of free variable access instead.

## Viewing Frames From a Stack

After displaying a particular stack with the L or control-L commands, the following commands view individual frames from that stack:

F *number* Prints the contents of frame *number*, where *number* is the number next to the frame name in the back trace.

Note: Unlike most other commands, *number* is in decimal.

The frame is printed in two parts, a basic frame containing the function's arguments and a frame extension containing control information, the function's local (PROG) variables, and dynamic values. On the left side of the printout are the octal contents of each cell of the frame, with an interpretation, usually as a Lisp value, on the right.

line-feed or control-J Shows the next frame (closer to the root of the stack). Same as F *n* + 1, where *n* is the number of the frame most recently viewed. Immediately after an L or control-L command, *n* is zero, so line-feed views the first frame.

↑ Shows the previous frame. Same as F *n*–1.

D *symbol* Shows the definition cell for *symbol*. A definition cell containing all zeros denotes an undefined function. A definition cell whose left half is less than 400 denotes an interpreted definition; you

| | |
|---|---|
| | can use the V command (below) to have it printed as a Lisp expression. |
| A *symbol* | Shows the top-level value of *symbol*. |
| P *symbol* | Shows the property list of *symbol*. |
| C *symbol* | Prints (using PRINTCODE) the code definition for *symbol*. |
| V *hi lo* | Interprets the virtual address *hi, lo* as a Lisp value and attempts to print it. Virtual addresses appearing in stack frames are already interpreted for you by the F command, as are those in value cells (the A command) and property lists (the P command), but you may want to use the V command if you find a virtual address inside some other structure. |
| B *hi lo count* | Prints *count* words of the raw contents of the virtual memory starting at virtual address *hi, lo*. This is most useful for examining the contents of a datatype, which other commands simply print as its virtual address, i.e., in the form {type}#hi,lo. |
| _ *hi lo number* | Sets the contents of the word at virtual address *hi, lo* to be *number*. This command obviously should be used with care. |
| control-V *symbol atomicValue* | Sets the top-level value of symbol to be *atomicValue*, i.e., this is a remote SETTOPVAL. Only symbols and small integers are acceptable values to set. In addition, if the previous value was not a symbol or small integer, it is not reference counted correctly, so will not be garbage collected. |
| U | Displays the debuggee's screen on your own (just the screen bit map, not the cursor). Typing any character restores your own screen. If the debugee's screen is larger than the debugger's, then you'll see that portion of the screen that fits. You can move the image of the remote screen by pressing the left mouse button and dragging the image, much like an over-size icon. |
| control-Y | Enters the Old Interlisp Executive under TeleRaid, where you can evaluate arbitrary Lisp expressions or call some of the functions listed below to perform TeleRaid operations for which there is no command. |
| | Use the Interlisp Executive's OK command to exit and return to TeleRaid. |
| Q | Quits TeleRaid without affecting the debuggee. |
| control-N | Executes the CONTROL-N TeleRaid command in the debuggee, i.e., causes the debuggee to resume execution, and quits TeleRaid. This command should not be used unless you are sure that the debuggee is resumable. |

## Viewing the System Stack

The following commands are for use by experts in stack format. A stack address is a number in the default radix denoting where the object of interest starts.

| | |
|---|---|
| W *address* | Walks sequentially through the system stack (i.e., by stack address, not by control or access links) starting at *address*, showing the stack frame type and its name (for frame extensions). If *address* is not given, this command shows the |

entire user stack. For the READSYS function (see next section) the walk starts at zero, so it shows the system stack as well.

| | |
|---|---|
| control-F *address* | Prints the basic frame stored at *address*. |
| control-X *address* | Prints the frame extension stored at *address*. |
| S *address count* | Prints raw contents of the stack (as with the B command) starting at *address* for *count* words. |

## Functions for Saving Work

The following functions do not have corresponding TeleRaid commands, but may be useful to call in the executive obtained from the control-Y command. They can be used to try to patch a broken sysout back into shape, or at least to save some of the work out of a workstation in a maintenance panel halt. Further functions like these can be written using the functions described in the next section.

(VLOADFNS *FN*) [Function]

Reads the EXPR definition of *FN* from the remote environment and stores it locally on *FN*'s EXPR property. *FN* can be a single symbol or a list of symbols.

(VLOADVAR *VAR*) [Function]

Locally sets the variable *VAR* to be the remote top-level value of *VAR*.

(VSAVEWORK) [Function]

Attempts to figure out what has changed and not been saved in the remote environment by looking at CHANGEDFNSLST, CHANGEDVARSLST and the property lists of files on FILELST. For each changed function or variable, it asks you whether to save it, and if so, it uses VLOADFNS or VLOADVAR to fetch it. You can then save the functions or variables from the locally running Lisp.

VSAVEWORK does not know how to save records, properties, etc., although a knowledgeable programmer could use the functions described in the next section to extend VSAVEWORK.

(VUNSAVEDEF *FN*) [Function]

Attempts to do a remote UNSAVEDEF by going down the VGETPROPLIST of *FN*, looking for properties CODE, BROKEN, and ADVISED. If it finds one, it stores the corresponding code object in *FN*'s remote definition cell, and prints a message saying what it has done.

For example, if you've managed to break something that's used by the interpreter, and have thus gotten into a recursive break, you might be able to recover by VUNSAVEDEFing it, then doing a control-D on the remote machine.

(VYANKDEF *NEWSYMBOL OLDSYMBOL*) [Function]

Yanks the definition from function *OLDSYMBOL* and stores it into *NEWSYMBOL*. For example, (VYANKDEF 'PRINTBELLS 'NILL) turns off ringing of the bell in the remote environment.

Note: VUNSAVEDEF and VYANKDEF do not adjust reference counts, or interact correctly with BREAK and ADVISE. They should be thought of as emergency patches designed to get the system running long enough to save state and bail out. In particular, do not call UNBREAK or UNADVISE on a function that you have applied VUNSAVEDEF to, and do not alter or remove its CODE, BROKEN, or ADVISED property. Similarly, do not redefine the function *OLDSYMBOL* that you have yanked a definition from.

## Implementation

TeleRaid is implemented in two parts: ReadSys, which reads a remote system's virtual memory, and VRaid, the interactive debugger described above. The remote virtual memory can be either a workstation running a TeleRaid server or a sysout file. The functions inside TeleRaid look like normal Lisp functions, but they are designed to operate on the remote virtual memory, rather than the normal (local) virtual memory. The remote versions of functions normally begin with V.

In general, TeleRaid is not a facility for the casual user. It is mostly used by system implementors performing very low-level debugging. The set of functions described here is a partial list, intended to help the serious programmer who has some interest in doing this kind of debugging.

### Reading the Remote Vmem

(READSYS *FILE WRITEABLE*) [Function]

Opens the remote virtual memory *FILE*, which should be the name of a file in sysout format. If *WRITEABLE* is T, then the file is opened for write access, so that commands that alter the virtual memory (e.g., the _ and control-V commands) are permitted. The main use for this is to patch sysouts in simple ways (e.g., by changing a global flag from NIL to T).

*FILE* can also be a list of one element, the PUP address of a machine running a TeleRaid server, in the same form as the *HOST* argument to the function TELERAID. In this case, *WRITEABLE* is ignored.

If *FILE* is NIL, READSYS closes any open virtual memory file, clears its data structures and reverts to examining no virtual memory.

(VRAID *RAIDIX*) [Function]

Runs the TeleRaid interactive debugger on the virtual memory most recently opened by READSYS.

## Manipulating the Remote VMem

The functions and macros described below directly manipulate the remote virtual memory. You can call them directly in the Lisp executive that you get by using the control-Y command under TeleRaid, or at the top level after calling READSYS. You can also, of course, write your own programs to use these functions. In order to use any of the macros below, you must LOADFROM the source file VMEM.

In the following functions, a pointer means a pointer into the remote virtual memory (the argument *PTR*), a 24-bit integer. All other arguments refer to local objects. Functions that fetch out of or store into the remote virtual memory operate on pointers. You can create a local copy of the structure denoted by a pointer by calling V\UNCOPY. You cannot do the inverse, i.e, create remote copies of local structures—the only local objects that you can translate into the remote virtual memory are symbols (assuming the same symbol exists remotely) and small integers.

Note: The functions that store into the remote memory should be used with care. None of these functions perform the proper reference counting. Therefore, if you are storing a value that ought to be reference-counted (roughly speaking, anything other than a symbol or small integer) and/or overwriting such a value, the garbage collector may get confused when the remote memory is resumed.

(VVAG2 *HI LO*) [Function]

Returns a pointer with hi-loc (top 8 bits) *HI* and lo-loc (low 16 bits) *LO*.

(VHILOC *PTR*) [Macro]

Returns the high part of *PTR*, i.e., (LRSH *PTR* 16).

(VLOLOC *PTR*) [Macro]

Returns the low part of *PTR*, i.e., (LOGAND *PTR* 177777Q).

(VADDBASE *PTR D*) [Macro]

Remote \ADDBASE: Returns a pointer that is *D* words beyond *PTR*, i.e., (IPLUS *PTR D*).

(V\UNCOPY *PTR*) [Function]

Returns a local copy of the remote structure pointed to by *PTR*. \UNCOPY only knows how to copy ordinary structures: symbols, integers (not bignums), floating-point numbers, characters, strings and lists. All other pointers, either as the argument to V\UNCOPY or inside structures copied by V\UNCOPY, are converted to local objects of type REMOTEPOINTER that print in the way that datatypes conventionally print—their contents are *not* copied.

(V\COPY *X*) [Function]

Returns a remote pointer to the local object *X*, which must be a symbol or small integer.

(VGETTOPVAL *ATOM*) [Function]

Returns a pointer to *ATOM*'s top-level value.

(VGETVAL *ATOM*) [Function]

Returns a local copy of *ATOM*'s top-level value, i.e., (V\UNCOPY (VGETTOPVAL *ATOM*)).

(VSETTOPVAL *ATOM VAL*) [Function]

Sets *ATOM*'s top-level value to be *VAL*, which must be a symbol or small integer.

(VGETPROPLIST *ATOM*) [Function]

Returns a pointer to ATOM's property list.

(VGETDEFN *ATOM*) [Function]

Returns a pointer to *ATOM*'s function definition.

(VTYPENAME *PTR*) [Function]

Returns the type name of *PTR*.

(VGETBASE0 *PTR*) [Function]

The most primitive fetching function: Returns the 16-bit integer stored in location *PTR*.

(VPUTBASE0 *PTR VAL*) [Function]

The most primitive storing function: Stores the 16-bit integer *VAL* into location *PTR*.

(VFIND.PACKAGE *NAME*) [Function]

Like the CL:FIND-PACKAGE, but returns the remote address of the named package or NIL if not found.

(VFIND.SYMBOL *NAME REMOTE-PACKAGE*) [Function]

Like the CL:FIND-SYMBOL, but returns the remote address of the named symbol.

(VGETBASE *PTR D*) [Macro]
(VPUTBASE *PTR D*) [Macro]
(VGETBASEBYTE *PTR D*) [Macro]
(VGETBASEPTR *PTR D*) [Macro]
(VPUTBASEPTR *PTR D VAL*) [Macro]

These are remote versions of \GETBASE, \PUTBASE, \GETBASEBYTE, \GETBASEPTR and \PUTBASEPTR, respectively. They are implemented in terms of VGETBASE0 and VPUTBASE0.

## Limitations

TeleRaid uses PUP, thus the machine being examined must be on the same network or reachable via PUP gateways.

This code has one major shortcoming which will not normally turn up. If the local and remote sysouts conflict in their package setups, it is possible for this code to return symbols interned in what for the Teleraiding machine would be the correct package, but for the remote machine is in fact incorrect. The problem lies in the fact that you cannot uncopy a symbol correctly between two machines with incompatible package setups. An example of such a situation would be where on one machine the package FOO inherits BAR, and on the other BAR is present directly in FOO. BAR's package cell will be different in the two cases.

[This page intentionally left blank]

# TEXEC

TExec is a version of the Interlisp-D executive which includes certain features of TEdit, so that commands can be edited, much like text. TExec preserves all of the functionality of the "old" executive (including history commands, ?=, DWIM, Programmer's Assistant, editing of the current input form, parenthesis matching/blinking, etc.) plus the ability to scroll anywhere in the output for viewing and/or copy-selecting old text.

TExec makes it easy to use Interlisp to get information, then use that information to build new commands to Interlisp.

TExec has two major advantages:

You can put into a window something longer than a windowful and still be able to scroll back and forth in it. In the regular exec window, all you see are the last few lines.

You can print something to the window, then use all or part if it at your next type-in.

## Requirements

TEdit

## Installation

Load TEDIT.LCOM and TEXEC.LCOM modules from the library.

## User Interface

The Executive is described in the *IRM* and in the *Lisp Release Notes*.

TEdit is described in the *Lisp Documentation Tools* manual.

## Starting TExec

TExec can be invoked interactively from the right-button (background) menu, or programmatically by calling

(TEXEC *REGION PROMPT MENUFN*) [Function]

If *REGION* is not specified, the system issues a prompt to create a window. If prompt is not supplied, a # is used as the prompt.

If *MENUFN* is not supplied, a command menu similar to TEdit is used. See the TEdit section in the *Lisp Documentation Tools* manual titled "Using the TEdit Window."

## Differences between TExec and TEdit

The following TEdit commands are not included in the TExec main menu: LOOKS, SUBSTITUTE, QUIT, AND EXPANDED MENU.

TExec has two Find commands which are not in TEdit: FORWARD FIND and BACKWARD FIND.

FORWARD FIND searches forward from the beginning of the text stream if no previous text string has been found or if the caret is in the current/next type-in; otherwise the search continues forward from the last find.

BACKWARD FIND searces backwards from the type-in point if it is the first time, or from the last place it found the text. You can force BACKWARD FIND to start from the type-in point by placing the caret there with the mouse.

To allow the easy copy-selection of entire lines of input, use a carriage-return/line feed as the prompt, and the prompt will be printed on a different line from the type-in; e.g., (TEXEC REGION "<CR><LF>").

Pressing the escape key does not cause recognition of keywords in USERWORDS as it does under TTYIN. The ↑R (retype input) and case-changing commands of TTYIN are not implemented. Display stream graphics are not saved in the output.

## Using TExec

TExec allows editing the current type-in using TEdit commands (see the TEdit section in the *Lisp Documentation Tools* manual titled "Editing Text"). Type-in is considered editable until a final matching right parenthesis, right bracket, or carriage return is typed, at which point it becomes immutable. Any output to a TExec window such as from CONTROL-T or ?= is placed in front of the current type-in so as not to interfere with your typing.

Unechoed input mode is implemented using a feature of TEdit known as invisible characters. Such characters, though invisible, are present in the buffer, and will be copied if they are within the bounds of a copy-selection. The primary terminal table, \PRIMTERMTABLE (the value of (GETTERMTABLE)) is used (different from TEdit) to allow control characters to be echoed as CONTROL-X (where is x is the control character), as they are in the Old Interlisp Executive.

The contents of a TExec window are saved in memory as a text stream. The maximum number of characters to be saved is specified by selecting the LIMIT command in the menu. When this limit is reached, characters are deleted from the beginning of the buffer as new ones are added to the end. The initial setting is 10,000 characters.

The escape key works the way it is described in the Programmer's Assistant section of the *IRM.* It is used as a character substitution mark by the Programmer's Assistant USE command.

## Limitations

TExec does not understand Common Lisp syntax, so it is best to call it from an Interlisp exec.

= ? is not implemented.

[This page intentionally left blank]

# TEXTMODULES

TEXTMODULES converts source code files from File Manager format to Common Lisp style plain text and back again. When exporting to plain text, a small number of File Manager coms types are supported. When importing from a plain text file, several convenience features are available including comment upgrade and conversion of specific named defmacros into defdefiners.

**NOTE:** The Text File Translator changes source code format only; this is **not** an Interlisp to Common Lisp translator.

All symbols described in this section are in the TEXTMODULES package, nicknamed TM.

This section describes the load and make processes, and the static format of text files and their File Manager counterparts. The File Manager counterparts are discussed in increasing detail until their programmability is covered.

## Overview

The Text File Translator supports the development of portable Common Lisp source code in the Lisp Environment. It brings portable Common Lisp sources into the File Manager without losing any of their contents. It also makes new text files based on the File Manager's "filecoms."

The original file's function and ordering are retained, but exact formatting is not. The pretty printer causes all comments and expressions on the text file to be uniformly formatted.

Exporting a source file into text and back again will lose grouping of definitions under their coms.

## Installation

Load TEXTMODULES.DFASL from the library.

## Dependencies

Special support for editing and printing of comments is required. This are provided by the SEDIT-COMMONLISP file. Some caveats on the editing of presentations are mentioned below.

The support file is automatically loaded by the TEXTMODULES file. SEDIT-COMMONLISP cannot operate without TEXTMODULES and must be loaded by it or after it.

File Manager source files created with **load-textmodule** depend on having TEXTMODULES and SEDIT-COMMONLISP loaded.

## Programmer's Interface

**load-textmodule** *pathname* &key *module install package upgrade-comment-length join-comments convert-loaded-files defdefiner-macros*

[Function]

Like **lisp:load**; the file indicated by *pathname* is loaded, but in addition filecoms ar created and other information is stored for the File Manager. Key arguments are described below.

(See below under **Text File Format** for a description of the format of text files which can be read by **load-textmodule**).

Local bindings of reader affecting variables are established and set to Common Lisp defaults, except for the readtable.

A special readtable is used which creates internal representations for objects normally lost during reading (see below under **Presentation types**).

If there are some simple forms to set up the read environment at the front of the file, they are recognized and moved into a newly created makefile environment (see below under **Makefile Environment** for a complete description of this).

Each form is read from the file (one at a time). If the form is recognized a description of it is given to the File Manager and its definition is installed. If the form is not recognized it is wrapped in a "top level form" filecom and then installed by stripping presentation objects and evaluating.

**defun** in a **let** at top-level is treated like any top-level form. Such forms should be edited directly in the filecoms. Not doing this can have curious consequences, since calling **ed** on the function name will not modify the definition in the **let** (which remains in the FILECOMS as a top level form).

No forms after the read environment forms should change the reader's environment.

When the file has been completely read its content description is given to the File Manager. Also added to the content description are properties declaring its il:filetype as :compile-file and makefile-environment as that of the text file (whether given by setting forms at the front of the file or by default).

Several key options are available:

*module* A string or symbol used to create the symbol used as the File Manager's name of this module. Strings have their case preserved. Symbols have their name strings taken. Defaults to the uppercased root name of the path.

*install* T or NIL. Indicates whether the definitions in the file should be installed in the running system. Any package setup makes it mandatory to install the definitions in a source file; e. g. since :INSTALL NIL means forms in the file are not evaluated, any IN-PACKAGE form would not be evaluated and the file would be read in the wrong package. This can sometimes be worked around using the :PACKAGE argument.

*package* A package name or package, defaults to "USER". This is the package the file will be read into.

*upgrade-comment-length* A number or NIL. Defaults to the value of ***upgrade-comment-length*** (which defaults to 40). The length, in characters, at which single semicolon comments are upgraded to double semicolon comments.

*join-comments* T or NIL. Defaults to the value of ***join-comments*** (which defaults to T). Causes comments of the same level in the coms to be joined together. This makes for more efficient editor operation, but loses all formatting inside of comments; e.g. inter-comment line breaks are not preserved.

*convert-loaded-files* T, :QUERY or NIL. Defaults to the value of ***convert-loaded-files*** (which defaults to :QUERY). If a REQUIRE or LOAD statement is noticed at top level a recursive call to LOAD-TEXTMODULE will be made. With :QUERY turned on the user is first prompted. If the pathname specified in the LOAD or REQUIRE is computed based on variables in the file being loaded :INSTALL must be true. Complex systems that contain special loading functions will not be handled by this mechanism.

*defdefiner-macros* A list of defmacro names. Defaults to the value of ***defdefiner-macros*** (which defaults to NIL). If a top-level defmacro is found whose name is on this list, the defmacro will be translated into an IL:FUNCTIONS defdefiner form. The defdefiner form then creates a macro that builds definers. Definers are the basic definition units maintained by the File Manager. DEFUN is itself a defdefiner macro. A particular DEFUN form is a definer for the named function (see the Lisp Release Notes, 4. Changes to Interlisp-D in Lyric/Medley, Section 17.8.2 Defining New File Manager Types, for more information on the defdefiner form).

Warning: names on this list must be in the correct package, i.e. the one the file will be read in. A typical way to use this feature is:

- Examine the text source file for DEFMACRO forms that are used to create defdefiners.
- Make the package which the text file's IN-PACKAGE expression will later find.
- Do LOAD-TEXTMODULES giving the :DEFDEFINER-MACROS key argument a list of fully package qualified symbols naming the defdefiners contained in the file.

**make-textmodule** *module* &key *type pathname filecoms width* [Function]

The File Manager's description of the file *module* is used to create a text file. *module* may be provided as either a string or a symbol. A string's case will be preserved. A symbol's name string is used. Keyword arguments are described below.

(See below under **File manager description of contents** for a description of filecoms that can be written out by **make-textmodule**.)

Local bindings of printer affecting variables are established and set to Common Lisp defaults, except for the readtable.

The file's environment is written out, based on its makefile-environment property (see below under **File manager source file format** for ways of expressing the environment).

The specially made description of the file's contents (from the File Manager) is iterated over to write out each form in the file.

Several key options are availible:

| | |
|---|---|
| *type* | A string, defaults to ".LISP". The file type extension to be used on the text file being written. |
| *pathname* | A pathname, defaults to the module name merged with the extension and ***default-pathname-defaults***. The file which will contain the new text file. |
| *filecoms* | A list of file commands may be supplied here. Defaults to the commands for the File Manager file named by module. |
| *width* | A positive integer, defaults to 80. The width, in characters, of lines in the text file. Used by the prettyprinting routines for formatting. |

## Variables that control loading

***join-comments*** [Variable]

T or NIL. Defaults to T. Causes comments of the same level in the file coms to be joined together. This makes for more efficient editor operation, but loses any formatting inside of comments, e.g. inter-comment line breaks are not preserved.

***convert-loaded-files*** [Variable]

NIL, :QUERY or T. Defaults to :QUERY. Controls whether a LOAD or REQUIRE statement at top level in a loaded text file will cause the referred to file to be recursively load-textmodule'd. If the pathname specified in the LOAD or REQUIRE is computed based on variables in the file being loaded the :INSTALL argument to **load-textmodule** must be true.

***upgrade-semicolon-comments*** [Variable]

NIL or a positive integer. Defaults to 40. Controls whether and at what length (in characters) a single semicolon comment is upgraded to a double semicolon comment. NIL inhibits upgrading.

***defdefiner-macros*** [Variable]

A list of defmacro names. Defaults to NIL. If a top-level defmacro is found by LOAD-TEXTMODULES whose name is on this list, the defmacro will be translated into an IL:FUNCTIONS defdefiner form. The defdefiner form creates a macro that builds definers. Definers are the basic definition units maintained by the File Manager. DEFUN is itself a defdefiner macro. A particular DEFUN form is a definer for the named defun (see the Lisp Release Notes, 4. Changes to Interlisp-D in Lyric/Medley, Section 17.8.2 Defining New File Manager Types, for more information on the defdefiner form).

Warning: names on this list must be in the correct package, i.e. the one the file will be read in. A typical way to use this feature is:

- Examine the text source file for DEFMACRO forms that are used to create defdefiners.
- Make the package which the file's IN-PACKAGE expression will later find.
- Do LOAD-TEXTMODULES giving the :DEFDEFINER-MACROS keyword argument a list of fully package qualified symbols naming the defdefiners contained in the file.

## Text file format

TextModules creates and understands the format of portable pure Common Lisp text files with very simple and constrained package setup information. The overall form of these files is described here as a guide to what sort of files may be imported.

An EMACS style mode line comment may optionally be present as the file's first item. It corresponds to the makefile-environment in the file manager.

```
; -*- Mode: LISP; Package: (FOO GLOBAL 1000); Base:10 -*-
```

| | |
|---|---|
| mode | For some versions of the EMACs editor this will declare the major mode, which arranges key to command bindings for LISP instead of documents. |
| package | Name, used packages and initial space for symbols. |
| base | Numeric "ibase" |

The mode line is generated by TextModules and is provided purely as a convenience in transporting code to EMACS based environments. It has no effect on the File Manager. The makefile-environment is actually instated using expressions directly following the mode line.

The Common Lisp community has agreed that portable text files will use only one reader environment and hence not switch packages or alter the readtable partway through. TextModules assumes that the reader environment is set up by the seven (plus two) standard environment modifying forms. These forms are recognized by the TextModules parser only if they appear at the front of the file and in order (comments being ignored):

| | |
|---|---|
| Put | **provide** |
| In | **in-package** |
| Seven | **shadow** |
| Extremely | **export** |
| Random | **require** (or **il:filesload)** |
| User | **use-package** |
| Interface | **import** |
| ... | **shadowing-import** |
| ... | **setf *read-base*** |
| Commands | Contents of module |

Portable files may optionally add ***read-base*** setting and **shadowing-import** expressions. Also, **il:filesload** may be used in place of **require,** when a Lisp file (not containing a **provide** form) must be loaded.

Any one of these forms is optional, but they must appear first in the file (and in order) to be parsed into a makefile-environment when **load-textmodule** is called.

*Warning*: this software applies heuristics to the package forms to make them independent of the package environment they are read in. These may not work and it is recommended that the makefile-environment be checked for correctness after load-textmodule brings in the file. Complex package setups will almost certainly not be handled correctly and should be created in a separate file which the main text file can REQUIRE.

The contents of a text file are a sequence of forms. Certain forms are understood by the File Manager and hence specially recognized. These recognized forms include:

| | |
|---|---|
| *Comments* | which are translated into Interlisp style comments |
| *Definers* | such as defun or defvar |
| *eval-when* | which is translated into a File Manager eval-when |
| *Read time conditionals* | which may become "unread objects" |

All other forms are considered *top level forms* and simply saved as is.

Definers hold onto presentations, e.g., read time conditionals, as well as comments. Comments and presentations are always available to be edited.

To see if something is a definer form, examine the property list of its name (like **defun**). Use the Exec's **pl** (print property list) command to look for the property **:definer-for**.

Note that only the above kinds of forms will be recognized by the TextModules parser on a portable Common Lisp file.

There are a few somewhat common problems that can arise when importing a text file. Chief among these are "bootstrapping definitions" and circularity.

## Problems to be aware of

Occasionally, when starting up a complex software system, it is useful to install a temporary definition until the mechanism required by the actual definition is in place. This can cause a definition by the same name to appear in more than one place in a text file. The Textmodules system will simply use the latter definition in the file, causing the next loading of it to fail for lack of the lost bootstrap definition. It is recommended that bootstrap definitions be made into "top level forms", e.g. a DEFUN can become a (SETF (SYMBOL-FUNCTION <name>) ...) form, DEFPARAMETER can become (SETF <name> ...), etc.

Also, some styles of programming may encourage creation of circular structure. Textmodules must map over top level forms to install presentations that contain comments, etc. Circular structures can cause these routines to fill memory with list structure.

## File Manager source files

Unlike standard Common Lisp, the File Manager is designed to keep all of a file's contents resident in memory as structure, rather than text. This scheme allows very fast update and editing of definitions. To maintain its own source files the File Manager keeps descriptions of the format (makefile-environment) and contents (filecoms) of a file.

The makefile-environment of the file is used to note the readtable and package the rest of the file to be read and printed in, and any file dependencies.

The filecoms maintain a description of the top-level defining forms in a file, like function and variable definitions. They also store plain top-level forms. Within any form there can appear lisp data, like vectors or "number in a radix." How these are read and printed are controlled by presentation types (described separately; see below).

It is important to separate the environment of the file from its contents because the File Manager (not TextModules) first reads all the forms in the file, and then evaluates them. Text based source files sometimes change the package as needed. This cannot work for the File Manager since the file's forms are all read and then executed, i.e. the package changes would not occur until after the entire file had been read, and forms after any IN-PACKAGE form would have been read incorrectly.

The File Manager first reads the makefile-environment forms in a well known environment (INTERLISP package, INTERLISP readtable) evaluates them to find the environment of the rest of the source file, then reads the rest of the source file in that environment. This is why the package environment setup forms are so carefully parsed out of text files being imported into the environment.

File Manager source files created by **load-textmodule** depend on both the TEXTMODULES & SEDIT-COMMONLISP modules. These must be loaded before source files created with them can be reloaded.

## File Manager source file format

The makefile-environment of a "managed" source file is used to control both how the exported text file and managed source files are printed. It is kept in a property named **il:makefile-environment** on the symbol with the root name of the file (in the INTERLISP package). This property is automatically generated when a portable text file is imported. The property is itself a plist containing :readtable, :package and :base values. The readtable used is called "LISP-FILE", a readtable defined by the TextModules program (hence File Manager source files created by **load-textmodules** depend on TextModules). The part of the makefile-environment of main interest is the :package. It sets the package in which the exported text file is printed.

Three forms of the makefile-environment's :package property are recognized.

- a string or symbol naming a package
- a **defpackage** statement
- a **let** statement

A string or symbol is simply taken to name a package.

A **defpackage** statement will have its portable components translated into a **let** statement as described below.

A **let** used for the makefile-environment should bind ***package*** and contain some form of the standard seven package and module setup forms (See above under "Text file format"). It should finally return the altered value of *package*. For example:

```
(let ((*package* *package*))
  ...environment setup forms.
  *package*
)
```

The forms in this expression must be written in a standard, pre-existing package, such as USER or XCL-USER. This is to break the circularity of writing a package defining form in the package it defines.

The package defining expressions in the let should follow all of the rules for portable text files (See above under "Text file format"), e.g., they should appear in order.

## File Manager description of contents

When a portable text file is imported its contents are parsed to produce the File Manager's filecoms (File Commands). File commands are a high-level way of viewing and controlling the ordering of definitions in a file. The following describes the filecoms produced when a text file is imported.

Forms on the file are either recognized as top level defining forms or wrapped in a "top level form" filecom. Several forms are recognized by TextModules itself and others can be added (see below under "Making new specifiers"). The constructs that recognize filecoms, both to export and import plain text, are called *specifiers*.

The following are placed in the filecoms based on the parsed contents of the text file:

**(il:*** ***type string*****)** Contains a comment string. *type* is a symbol of one, two, three or four semicolons, or a vertical bar. This handles top level single, double or triple semicolon comments, as well as balanced comments. When viewed in SEdit these display in real comment format, instead of the internal list representation.

**(eval-when** ***when . filecoms*****)** Wrapper with an evaluation time and containing more filecoms.

*definers* All definers are recognized, e.g. DEFUN, DEFMACRO, DEFVAR, DEFPARAMETER, DEFSTRUCT, etc. The definer specifier also converts DEFMACRO forms on the ***defdefiner-macros*** list to defdefiners during loading, and vice versa on printed to a text file.

**(il:p (top-level-form** ***form*****))** Top level form wrapper with a macro that calls the presentation translator. This filecom contains expressions which were not recognized and must be evaluated at load time. This kind of filecom also handles top level occurances of conditional read and read time evaluations (hash comma and hash dot). The top-level-form specifier also looks for LOADed or REQUIREd files and, depending on the variable ***convert-loaded-files***, attempts to convert the loaded files as well.

When the file is loaded and before evaluating these forms any presentation objects in them are stripped out (as for comments) or installed (as for read time evaluations). This is done by the TOP-LEVEL-FORM macro, which dispatches to the translation functions for the particular presentation objects. i.e., this allows comments to appear anywhere in the forms and not affect evaluation.

The above coms are created when a text file is imported. There are also a few specifiers provided to export filecoms, but not create them on import. These are convenient for exporting typical File Manager files. They are:

**(il:coms .** ***filecoms*****)** Used to group together definitions in a File Manager source file. The *filecoms* are dumped onto the text file in order. No information is placed on the resulting text file to preserve the grouping of the filecoms; if the exported text file is later imported the coms grouping will not reappear.

**(il:vars** . *descriptors***)** As described in the *IRM*. The *descriptors* are exported as **defparameter** forms. If the exported text file is later imported these **vars** coms will reappear under **variables** coms.

**(il:initvars** . *descriptors***)** As described in the *IRM*. The *descriptors* are exported as **defvar** forms. If the exported text file is later imported these **initvars** coms will reappear under **variables** coms.

**(il:constants** . *descriptors***)** As described in the *IRM*. The *descriptors* are exported as **defconstant** forms. If the exported text file is later imported these **constants** coms will reappear under **variables** coms.

**(il:props** . *descriptors***)** As described in the *IRM*. The *descriptors* are exported as **(setf (getf ...) ...)** forms. If the exported text file is later imported these **props** coms will reappear under **p** coms (top-level forms).

**(il:prop** *props* . *symbols***)** As described in the *IRM*. The *props* and *symbols* descriptors are used to generate forms for export, e.g. **(setf (getf 'foo 'bar) 21)**. If the exported text file is later imported these **prop** coms will reappear under **p** coms (top-level forms).

**(il:files** . *items***)** As described in the *IRM*. The *items* are used to generate forms for export, e.g. **(load "Foo.lisp")**. All options except **noerror** are ignored, the latter will cause the **:if-does-not-exist nil** key argument to be included in the load expression. If the exported text file is later imported these **files** coms will reappear under **p** coms (top-level forms).

## Making new specifiers

Specifiers are the glue that relate forms on a plain text file to filecoms in a File Manager source file. They can be considered addenda to the filepkgtype mechanism of the File Manager.

***specifiers*** [Variable]

A list of specifiers (its default contents are described below). New specifiers should be added to this list. This list is searched linearly; its order is significant mostly in that the default top-level form recognizer must always be last.

**make-specifier** &key *name filecom-p form-p add-form install-form print-filecom* [Function]

This function creates new specifiers for inclusion on the ***specifiers*** list. A specifier maps between the forms in a text file's contents and filecoms. It is the basis for importing and exporting top-level forms. Specifiers can be nested, as for EVAL-WHEN.

To do all of this a specifer contains functions that recognize forms of its kind on the text file and coms in filecoms, as well as functions that add the definition to the filecoms and install the definition as the one to be used at runtime. Finally, there is a function which prints a form onto a text file based on a com on the fileoms.

*name* A string naming the specifier.

*filecom-p* Predicate on FILECOM which returns true if it is the one used to represent this specifier in the filecoms of a managed file.

*form-p* Predicate on FORM, a form read from a text file being imported, which returns true if this is the specifier for the definition in FORM.

*add-form* Function of FORM and FILECOMS. FORM is a form read from a text file being imported, and which has already been confirmed with the *form-p* method. Should make the definition in FORM available (editable) in the programming environment (File Manager). It should make the definition editable and add a filecom for FORM to the FILECOMS description. It should return the new FILECOMS description. To *add-form* runs of subforms use **add-form** and **form-specifier** (see below).

Care should be used when making a definition editable. The simplest instance of this occurs when the FORM's definition is a definer. In this case its evaluation may be wrapped in a binding of **il:dfnflg** to **il:prop** to ensure that the definition form goes into the table of current definitions *without being evaluated*.

Adding a filecom to the FILECOMS should be done in a way that preserves ordering. The simplest way to do this is to append the new filecom to the end of the current FILECOMS.

*install-form* Function of a FORM which makes the definition in FORM the current one to be used in execution. If the defining mode flag indicates that the file is being loaded for editing only this function *will not be called* during loading of the form (il:dfnflg is set by the :install option to load-textmodule). To *install* runs of subforms use **install-form** and **form-specifier** (see below).

Care should be used when making a definition executable. The simplest instance of this occurs when the FORM's definition is a definer. In this case its evaluation may be wrapped in a binding of **il:dfnflg** to **t** to ensure that the definition form *is actually evaluated*.

*print-filecom* Function of FILECOM and STREAM which should generate a new line and pretty print a form, representing the FILECOM, onto the stream. To print runs of subforms use **print-filecom** and **filecom-specifier** (see below).

The semantics of the *add-form* and *install-form* methods remove some confusion between loading a definition into memory for editing (loading PROP) and installing that definition as the currently executable one (loading T). The *add-form* method makes the definition editable and the *install-form* makes it executable.

The following functions are used to handle subforms EG. In the eval-when specifier there are subforms that need to be parsed.

**form-specifier** *form* [Function]

Searches the current list of specifiers in an attempt to recognize *form* (as read from a text file being imported). Returns a specifier for *form*. If none is found a warning is signalled and a "do nothing" specifier is returned.

**filecom-specifier** *filecom* [Function]

Searches the current list of specifiers in an attempt to recognize *filecom* (as found in the filecoms of the managed file). Returns a specifier for *filecom*. If none is found a warning is signalled and a "do nothing" specifier is returned.

**add-form** *form filecoms &optional specifier* [Function]

Adds the *form* to the *filecoms* description based on the add method in the *specifier*. If *specifier* is not provided **form-specifier** will be used to get it from *form*. Returns the new filecoms. **nil** is used as an empty contents description.

**install-form** *form &optional specifier* [Function]

If the current definition mode (**il:dfnflg**) allows it, installs the *form* as current and executable based on the *specifier*. If *specifier* is not provided **form-specifier** will be used to get it from *form*.

**print-filecom** *filecom stream* &optional *specifier* [Function]

Prints a new line on *stream* and then pretty prints a form representing the filecom onto the *stream*. If *specifier* is not provided **filecom-specifier** will be used to get it from *filecom*.

## Presentation objects

Presentation objects represent things that normally disappear during reading, like comments or numbers written in a particular base. Each presentation object must be capable of being read from a text file, edited with SEdit, installed as it would be when normally read, and printed to a text file in its original form.

### Presentations supported by Lisp

Many presentations are already supported by SEdit :

| | |
|---|---|
| *#\character* | Character object. |
| *#:symbol* | Uninterned symbol. |
| *#'function* | Hash quote function abbreviation. |
| *; comment* | |
| *;; comment* | |
| *;;; comment* | |
| *;;;; comment* | Semicolon comments. Internal formatting is preserved when these are imported, including CRs and tabs, etc. Adjacent comments are *not* smashed together so that line breaks are preserved. A single leading space in a comment is ignored, since comments are always printed with a single leading space. |

These comment types are represented internally in the same way as in Lisp, i.e., a list beginning with the symbol IL:*, following by a symbol (interned in the INTERLISP package) containing one through four semicolons.

*#|comment|#* A balanced comment. Imported and exported by TextModules. Directly supported by SEdit as though it were a "level 5" semicolon comment.

This comment type is represented internally in a manner similar to semicolon comments, but where the symbol containing semicolons is replaced by a symbol whose name is the vertical bar character.

## Presentations supported by SEDIT-COMMONLISP

Several presentations are specially supported by SEDIT-COMMONLISP. Any of these can be created using the following commands in SEdit:

| | | |
|---|---|---|
| Read time conditionals | Control-N | Hash minus |
| | Control-P | Hash plus |
| Read time evaluation | Control-Q | Hash dot |
| Load time evaluation | Control-F | Hash comma |
| Octal notation | Control-I | Hash "O" |
| Hexidecimal notation | Control-J | Hash "X" |
| Binary notation | Control-K | Hash "B" |

*# + feature form*
*#-feature form* Read time conditionals.

A conditional expression can be either *unread* or *read* depending on whether the truth of its features expression and sign parse true (see *Common Lisp, the Language*). *Read* conditional expressions are stored as structure. *Unread* conditional expressions are stored as strings due to the potential inclusion of, e.g., numbers of higher precision, symbols in unknown packages, or the inclusion of unknown reader macros.

*Unread* read time conditionals are read by remembering file position, doing a read suppress read, backing up to the original position and saving all the characters between in a string. This means that streams from which conditional read presentations are read must be capable of random access (the TTY is not).

These are represented by hash-plus and hash-minus structures.

Editing of read time conditional presentations is not quite WYSIWYG. Feature symbols should always be given as keywords (this is done implicitly by the lisp reader, but not in SEdit) and unread forms appear in strings and must be edited as such.

*#.form* Read time evaluation. Hash dot is represented by the hash-dot presentation.

*#,form* Load time evaluation. Hash comma is represented by the hash-comma structure.

| | |
|---|---|
| *#**O**rational* | |
| *#**X**rational* | |
| *#**B**rational* | Rational representations (**O**ctal, he**X**idecimal, and **B**inary). These are represented by the hash-o, hash-x and hash-b structures. |

## Presentations not directly supported

It is not possible to directly edit the following presentations in SEdit:

| | |
|---|---|
| *#(contents)* | Vectors |
| *#rank*A*(contents)* | Arrays |
| #S(*name field1 value1 ...*) | Structures |
| #*1010101 | Bit vectors |

Any of these may be edited by opening an inspector from SEdit: selecting the object and using the Meta-E command on it. Any of these may be created in SEdit by inserting the appropriate **make-** expression, selecting it, and using the Meta-Z command with **cl:eval** as the mutating function.

## Presentations not supported

The following standard Common Lisp presentations are not supported by either SEdit or TextModules:

| | |
|---|---|
| *#n = object* and *#n#* | Object tag and reference notation |
| *#base*R*number* | Radix notation |

# VIRTUAL KEYBOARDS

VirtualKeyboards lets you change the behavior of your Lisp workstation keyboard to mimic another keyboard, hence making yours a "virtual" version of that other keyboard. It also lets you display pictures of keyboards on your screen and use them as menus for typing occasional special characters. Several keyboards may be displayed on the screen at once, letting you switch easily among keyboards for several languages and making hundreds of characters available for typing.

The virtual keyboards supplied with the module are Dvorak, German, Greek, Italian, logic, math, Spanish, European accents, and standard Russian. You can also define new keyboards with the associated Keyboard Editor module, which lets you edit a keyboard while seeing the actual look of the characters.

The virtual keyboards can be used with TEdit, DEdit, in the Lisp Executive window, for any application with which you use your keyboard.

## Requirements

You need one of the following files, as appropriate for the machine you are using:

DANDELIONKEYBOARDS
DORADOKEYBOARDS
DOVEKEYBOARDS

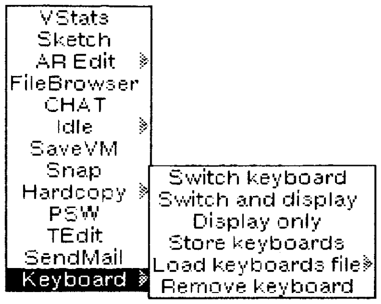
## Installation

Load VIRTUALKEYBOARDS.LCOM from the library. VirtualKeyboards loads the xxxKEYBOARDS files.

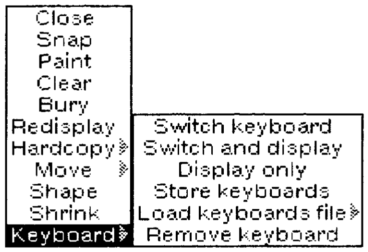
## User Interface

Loading VirtualKeyboards adds the item KEYBOARD to the background menu and the default window menu. Using the mouse in this module is the same as in the KeyboardEditor.

Selecting this item from the background menu changes your keyboard for all windows.



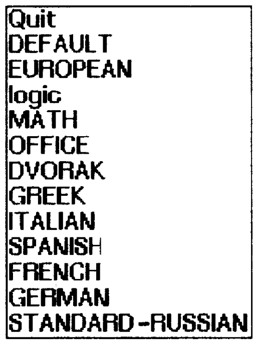
Selecting this item from the default window menu allows you to specify a keyboard for an individual window.



The main keyboard module commands are described in detail below.

## Switch Keyboards

Use the SWITCH KEYBOARD command to change the behavior of your keyboard to that of a selected virtual keyboard. This brings up a menu of the keyboards currently known to the program.



Select the keyboard you want to substitute for your workstation keyboard. Once you have changed your keyboard's behavior, pressing a key will send the character newly assigned to that key to the current input stream.

## Switch & Display a Keyboard

Use the SWITCH AND DISPLAY command to change the behavior of your keyboard to that of a selected keyboard and, in addition, display that keyboard's layout on the screen. You will be offered a menu of the keyboards known to the program; select the one you want to substitute for your workstation's keyboard. Displaying the keyboard layout helps if you're typing on an unfamiliar keyboard. SWITCH AND DISPLAY lets you type characters by using the keyboard displayed on the screen as a menu.

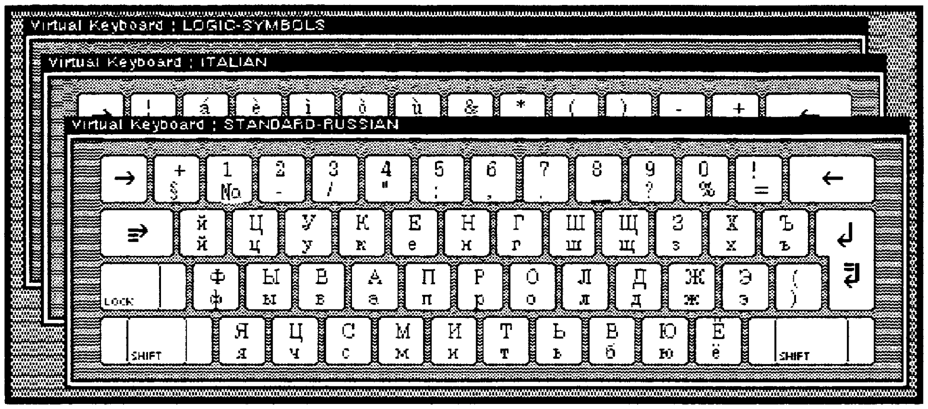

Virtual Keyboard : LOGIC-SYMBOLS
Virtual Keyboard : ITALIAN
Virtual Keyboard : STANDARD-RUSSIAN
LOCK
SHIFT
SHIFT

*Figure 9. Keyboard layout display*

## Display-Only a Keyboard

You can display the layout for any given virtual keyboard using the DISPLAY ONLY command. You will be offered a menu of the keyboards known to the program (such as above); select the one you want to display. This is useful if you are primarily using the standard English keyboard but need to type some characters in other languages, or some special characters such as mathematical symbols.

You can use the displayed image as a menu: Selecting a key from the image with the left mouse button will send the character assigned to that key, and pressing the shift key while you click on a key will send the shifted character. Middle-clicking also sends the shifted character.

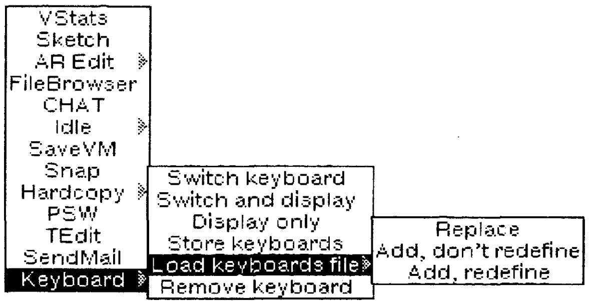
The effect is exactly as if you had pressed a key with that character assigned to it (except that interrupt characters are treated as ordinary characters; i.e., they do not cause an interrupt). The character is sent to the process that has the TTY (usually where the caret is flashing).

## Store Keyboards

After you edit a keyboard (using the KeyboardEditor module), you can store it using the STORE KEYBOARDS command in the top-level menu. When you select STORE KEYBOARDS, the system will prompt you for a file name. After you type the file name into the prompt window, the system will store all the keyboards known to it (both new and old) in that file in a form that will enable it to load them.

## Loading Keyboards File

To load a keyboards file, choose the LOAD KEYBOARDS FILE command, then slide the mouse cursor to the right and choose one of the three items in its submenu.



### Replace All Known Keyboards

Choosing REPLACE will load a set of keyboards that you stored using the StORE KEYBOARDS command, replacing all the keyboards currently known to the system.

The currently known keyboards will be lost.

### Add New Keyboards to the List of Known Keyboards

To add new keyboards without replacing any of the currently known keyboards, select ADD--DON'T REDEFINE. This will load a set of keyboard definitions.

If a keyboard in the file has the same name as one that is already known to the system, that keyboard will not be loaded and the current definition will stay in effect.

### Load New Keyboards and Redefine Existing Keyboards

The ADD--REDEFINE command is similar to ADD--DON'T REDEFINE, except that it redefines existing keyboards that have the same name as keyboards on the file.

Currently known keyboards that do not have the same name as newly loaded keyboards will remain in the list of known keyboards.

## Removing Keyboards From the Menu

To remove a keyboard from the set of currently known keyboards, select the REMOVE KEYBOARD command. This will pop up a menu of the known keyboards (such as above), from which you can select a keyboard to be deleted.

## Defining a Virtual Keyboard

A virtual keyboard is a list whose CAR is the name of the keyboard and whose CDR is a list of key actions. Creating a new virtual keyboard can be done directly in Lisp or interactively, using the KeyboardEditor module.

### Using the Functional Interface

The list of keyboards that are known to the program appears in the menu of keyboard names that pops up when you select SWITCH KEYBOARD from the background menu. This list is stored in the global variable VKBD.KNOWN-KEYBOARDS (see below). To add a keyboard to the list, you have to define that keyboard. To define a keyboard you can either call the function DEFINEKEYBOARD or manipulate the variable VKBD.KNOWN-KEYBOARDS directly as explained herein.

You may also use the KeyboardEditor module, which provides a menu-based user interface for creating and changing keyboard layouts.

A virtual keyboard is a list of the form

(KEYBOARD-NAME *KEY-ASSIGNMENT1 KEY-ASSIGNMENT2* . . .).

A *KEY-ASSIGNMENT* is a list of the form

(*KEY* (*UNSHIFTED-CHAR SHIFTED-CHAR LOCK/UNLOCK*)).

*KEY* is a key name (the character that appears on the actual keyboard).

*UNSHIFTED-CHAR* and *SHIFTED-CHAR* are character codes. Each can be either an integer representing the actual code or a list of two elements: the number of the character set and the number of the character in the set.

*LOCK/UNLOCK* is either the atom LOCKSHIFT, in which case *SHIFTED-CHAR* will be transmitted when the shift-lock key is down, or NOLOCKSHIFT, in which case the shift-lock key has no effect on that key. *LOCK/UNLOCK* is LOCKSHIFT by default.

(DEFINEKEYBOARD *KEYBOARD-NAME LIST-OF-KEY-ASSIGNMENTS KEYS-ARE-NUMBERS?*) [Function]

Creates a new virtual keyboard after parsing the list of key assignments and adds the keyboard to the list of known keyboards.

If *KEYS-ARE-NUMBERS?* is T, the function expects to find key numbers instead of key names.

(SWITCHKEYBOARDS *NEW-KEYBOARD SWITCH-FLG DISPLAY-FLG MENU-POSITION*) [Function]

Switches the current keyboard to *NEW-KEYBOARD*, where *NEW-KEYBOARD* is either a virtual keyboard or the name of a known keyboard.

If *SWITCH-FLG* is non-NIL, the actual key actions of the keyboard will be modified.

If *DISPLAY-FLG* is non-NIL, a window with a menu will be displayed. This displayed keyboard will act as a menu and will send characters to the current input stream when a character is selected.

VKBD.KNOWN-KEYBOARDS [Variable]

Contains the list of all currently known virtual keyboards.

## Limitations

After loading the Dvorak keyboard, and then restoring defaults, you lose the shift- lock key.

This is a new implementation of a facility similar to but not compatible with the Lyric library module WhereIs. Where-Is indexes all definers, but WhereIs only indexed Interlisp FNS definitions.

## Requirements

Hash-File and Cash-File.

## Installation

Load WHERE-IS.DFASL and the required .DFASL modules from the library.

## Changed File Manager Functions

Where-Is allows the file manager to know of many more definitions than are actually in the files which have been noticed. In order to achieve this behavior, the following file manager functions are changed when Where-Is is loaded.

Both of these functions are called by the edit interface (the function cl:ed). Thus when Where-Is is loaded the contents of its databases are known to the editor.

(`il:whereis` *name type files filter*) [Function]

Performs as described in the Interlisp Reference Manual. Returns the subset of *files* that contain a *type* definition for *name*. *Files* defaults to il:filelst (all noticed files). When Where-Is is loaded and il:whereis is passed t as its files argument il:whereis will look in the Where-Is databases.

(`il:typesof` *name possible-types impossible-types source filter*) [Function]

Performs as described in the Interlisp Reference Manual. Returns the subset of *possible-types* that *name* is defined as. *Possible-types* defaults to il:filepkgtypes (all define types). When Where-Is is loaded il:typesof will also include the types for *name* in its databases.

## Databases

Where-Is provides functions to use and build databases.

## Using a Database

`(xcl::add-where-is-database` *pathname*`)` [Function]

Adds the database in the file named by *pathname* to the databases known to Where-Is. If a database on an older version of this file is already known, then Where-Is will start using the new version.

`(xcl::del-where-is-database` *pathname*`)` [Function]

Deletes the database named by *pathname* from the databases known to Where-Is.

`xcl::*where-is-cash-files*` [Variable]

Contains the list of databases known to Where-Is.

There is a proceed case for errors while accessing a database which will delete the offending database. This can be useful when a file server goes down.

## Building a Database

`(xcl::where-is-notice` *database-file* `&key` *files new hash-file-size temp-file-name define-types quiet*`)` [Function]

Records the definers on *files* in the file named by *database-file*.

*Files* can be a pathname or a list of pathnames. The default for *files* is `"*.;"`. Note that it is important to include the trailing semi-colon so that only definers on the most recent version are recorded.

If *new* is true a new database file will be created, otherwise *database-file* is presumed to name an existing Where-Is database to be augmented. The default for *new* is `nil`.

*Hash-file-size* is only used when new is false and is passed as the *size* argument to make-hash-file. The default for *hash-file-size* is `xcl::*where-is-hash-file-size*`, which has a default top-level value of 10,000.

If *temp-file-name* is provided then all changes will happen in the temporary file named, which will afterwards be renamed to *database-file*. This can both make things faster (if the temporary file is on a faster device) and doesn't generate a new version of a database until the new version is ready to be used. The use of a temporary file may slow things down when a large existing database is just being updated to reflect a small number of changes.

*Define-types* is the list of define types (file package types) which should be recorded. The default define types are all those on IL:FILEPKGTYPES which are not aliases for others and which are not in the list `xcl::*where-is-ignore-define-types*`.

Unless *quiet* is true, `xcl::where-is-notice` will print the name of each file as it is processed.

`xcl::where-is-notice` returns the pathname of the hash file written.

[This page intentionally left blank]

# 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).

## STARTING WHO-LINE

Loading Who-Line.dfasl from any Executive will automatically open the Who-Line at the bottom of the display. If you close the Who-Line window, it can be reopened with the function (INSTALL-WHO-LINE-OPTIONS) described below

### 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; and an optional form that will reset any internal state of the entry when evaluated.

[[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-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.

## 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 Xerox 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.

(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.

# INDEX

Envos Corporation

# READER COMMENT FORM

WE WOULD APPRECIATE YOUR COMMENTS AND SUGGESTIONS FOR IMPROVING THIS PUBLICATION.

| | | | |
|---|---|---|---|
| PUBLICATION NUMBER | RELEASE DATE | TITLE | CURRENT DATE |

HOW DID YOU USE THIS PUBLICATION?

☐ LEARNING ☐ WRITING/GRAPHIC

☐ REFERENCE ☐ INSTALLATION

IS THE MATERIAL PRESENTED IN THIS GUIDE:

☐ FULLY COVERED ☐ WELL ILLUSTRATED ☐ WELL ORGANIZED ☐ CLEAR

WHAT IS YOUR OVERALL RATING OF THIS PUBLICATION?

☐ VERY GOOD ☐ FAIR ☐ VERY POOR

☐ GOOD ☐ POOR

DO YOU HAVE SUGGESTED CONTENT CORRECTIONS, CHANGES OR ADDITIONS?

☐ YES ☐ NO

YOUR OTHER COMMENTS MAY BE ENTERED HERE. PLEASE BE SPECIFIC AND GIVE PAGE, PARAGRAPH AND LINE NUMBER REFERENCES WHERE APPLICABLE.

YOUR NAME & RETURN ADDRESS

C U T O N D O T T E D L I N E

THANK YOU FOR YOUR INTEREST (FOLD AND FASTEN AS SHOWN ON BACK AND MAIL)

TAPE HERE ONLY

FOLD

No Postage
Necessary
If Mailed
in the
United States

**BUSINESS REPLY MAIL**

First Class Permit No. 1744 Mountain View, California

Postage will be paid by Addressee

Envos Corporation
Attn: Customer Support
1157 San Antonio Road
Mountain View, California 94043

FOLD