Venue

Lisp Library Modules

Medley Release 2.0
August 1991

Address comments to:
Venue
User Documentation
1549 Industrial Road
San Carlos, CA 94070
415-508-9672

LISP LIBRARY MODULES

Medley Release 2.0

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

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 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 the following:

- General description of the module
- 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
- Samples 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 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 Titan`; their arguments are in *Classic ITALICS*.

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

A backslash (\) character preceding a function or variable name signifies that it is a system–internal name.

Examples are shown in `Titan`.

Messages displayed in the prompt window are shown in **`Titan 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, Second Edition, by Guy L. Steele Jr., Digital Press, 1990.

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 2.0 User's Guide, Medley Release, 1991.

[This page intentionally left blank]

INTRODUCTION

This information pertains to the library modules as a whole, as well as the interaction among modules. This section contains changes that have occurred in the library modules since the Lyric release. Changes for Medley Release 1.2 are indicated with revision bars in the right margin.

Primary change:

- The new Color module has been added at the end of the binder.

What You Should Look For

Before running the modules, pay particular attention to the file dependencies section below. Most modules use 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 Added to the Library

Color

Details of Changes

The following module was added to the library with the Medley Release 1.2.

Color

Now supports color Sun Workstations.

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	GRAPHER	MASTERSCOPE
CASH-FILE	HASH-FILE	
CHAT	DMCHAT CHATTERMINAL	PUPCHAT or NSCHAT or RS232CHAT or TTYCHAT 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	READNUMBER	SCALEBITMAP
FILEBROWSER	TABLEBROWSER	Printer drivers, fonts TEDIT, SEDIT, DEDIT
FONTSAMPLE		FONTSHEETx.IP
FX-80DRIVER		DLRS232C or DLTTY
GRAPHZOOM		

GRAPHER

HRULE	IMAGEOBJ EDITBITMAP TEDIT

KERMIT
 	
    KERMITMENU
        KERMIT
	CHAT
	RS232 or TCP protocols

KEYBOARDEDITOR
    VIRTUALKEYBOARDS
        see below	

MASTERSCOPE
    MSPARSE
    MSANALYZE
    MSCOMMON
    MS-PACKAGE
	BROWSER, DATABASEFNS, a Lisp editor

MINISERVE	NS or PUP or XNS
NSMAINTAIN DES	
PRESS PUPPRINT	FONT.WIDTHS

RS232
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
	KERMIT

- SPY
  - GRAPHER
  - READNUMBER
  - IMAGEOBJ
    - EDITBITMAP
      - see above

- SYSEDIT
  - EXPORTS.ALL
    - CMLARRAY-SUPPORT
- MASTERSCOPE

- 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

COLOR
- MAIKOCOLOR
- LLCOLOR
- TAKEBIGBM

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 modules are those of Interlisp–D, and therefore you will have to make sure that you use the IL: prefix when you are not in Interlisp .

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]

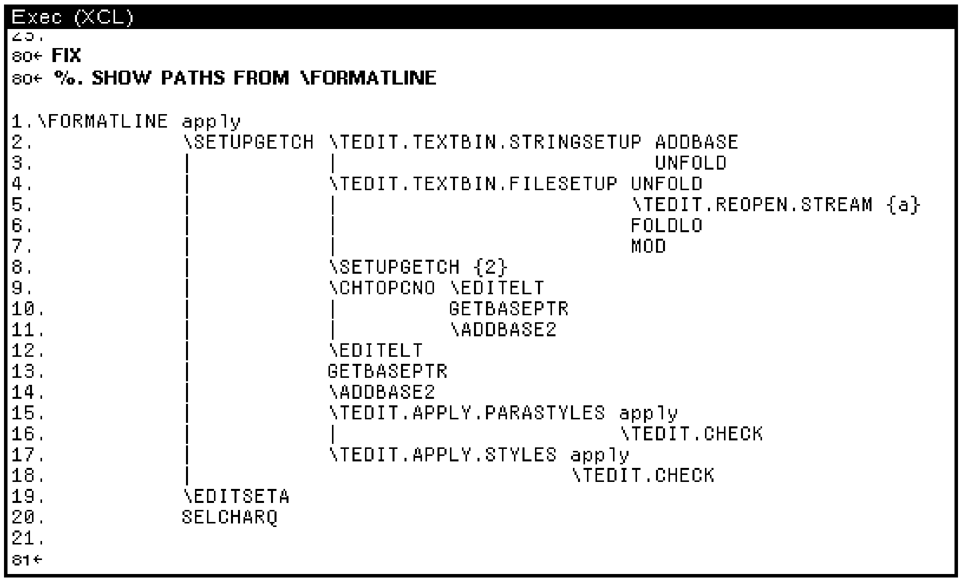
INDEX

BROWSER

4/26/2023 Larry Msinter: added BROWSERMAX to limit depth of tree, and made left-button on a node on the browser graph offer all formatting / control.

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

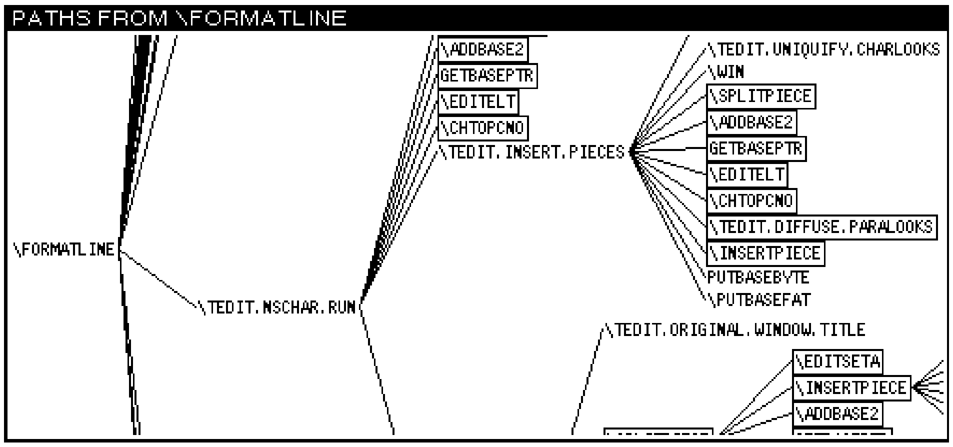
Browser makes it easier to use the MasterScope SHOW PATHS command by making it easier to read the result. This is how SHOW PATHS looks before Browser is loaded:



```
Exec (XCL)
23.
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:



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

You can reshape or scroll the window 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 using the mouse. Clicking with the left button over the name of a function causes a menu to appear with menu items: Display will pretty-printed in the Browser printout window. Describe will describes the function, using the MasterScope `DESCRIBE` command, in the Browser describe window. The Edit menu item will calls the editor on that function. There are two menu items that will show you a new graph rooted at the function selected: CallsFrom and CallsTo.

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

Browser Variables

`BROWSERFORMAT` [Variable]

Controls the layout of the graph. It is set initially to display the graph horizontally, with the root to the left, and links running left to right.

Example:

`BROWSERFORMAT=(VERTICAL REVERSE)`

Lays out the graph with the root at the bottom and links running from bottom to top. For more details, see the description of the `LAYOUTGRAPH` function in the GRAPHER Library Module.

`BROWSERMAX` [Variable]

Maxumum depth of the graph. Initially 6.

`BROWSERBOXING` [Variable]

Controls how duplicate nodes in the graph are displayed. Its initial setting boxes functions that occur more than once in the graph.

Example:

`BROWSERBOXING = ((MARK BORDER 5) COPIES/ONLY)`

Boxes copies of the function with a border of 5. For more details, see the description of the `LAYOUTGRAPH` function in the GRAPHER Library Module.

Functions

`(BROWSER T)` turns the Browser on.

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

`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 to 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 to which *cash-file* is a front end.

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.

[This page intentionally left blank]

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 `BOUT`ed to `CENTRONICS.STREAM` are sent to the attached printer. You may only have one stream open to the parallel port at one time; attempts to open others 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.

[This page intentionally left blank]

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 have not 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, 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 are prompted in the system's prompt window for the name of a host to which to connect. Subsequently, you are prompted with a menu of all hosts to which you have opened Chat connections; the last entry in this menu is `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 fails 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 are 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 is switched to that window. At the same time, keyboard interrupts, such as Control-E, are reenabled. Whenever you select an open Chat window, the focus-of-attention is returned to the Chat window, and keyboard interrupts are 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 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` only affects new Chat connections.

`CHAT.INTERRUPTS` [Variable]

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

Like `CHAT.KEYACTIONS`, this variable only affects new Chat connections.

`CHAT.ALLHOSTS` [Variable]

A list of host names, as uppercase symbols, to which you want 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: To work well with the DM2500 and VT100 terminal emulators, you should use fixed–width fonts (e.g., Gacha or Terminal).

`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.

Note: Before a new value of `CHAT.WINDOW.SIZE` is used, `CHAT.WINDOW` must be set to `NIL` or `NOBIND`. If `CHAT.WINDOW.SIZE` is changed after chat has already been called, and chat is then called, the window is not changed because the information is cached. `CHAT.WINDOW` must be set to `NIL` and the window recreated anew before this takes place.

`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 are prompted to choose a service from a menu of all known virtual service types. If this fails, NS Chat abandons 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 automatically chooses 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 are prompted for a telephone number; the ECS dials that number and completes 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 loads and initializes the TCP-IP module, if necessary. Read the TCP-IP module in this manual for more information.

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. Read the RS232 module in this manual for more information.

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 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]

CLIPBOARD

Written by Ron Kaplan, 2020-2021

A small package that implements copy and paste to the system clipboard.

It arms meta-C for copy to the clipboard from the current selection of an application that has been armed (TEDIT, SEDIT), and also meta-X (TEDIT) for extraction (copy followed by delete). (SEDIT currently assigns "expand" to meta-X).

Meta-V is defined as an interrupt character that pastes the current clipboard contents into whatever process curently has input focus.

The information in the clipboard can be provided from or to external (non-Medley) applications (mail, emacs, etc.) in the usual way. For example, a form selected in SEDIT can be copied to the clipboard and pasted into an Outlook or Mac Mail email message.

It assumes that the external format of the clipboard is determined by `(SYSTEM-EXTERNALFORMAT)`, and characters will be converted to and from the Medley internal character encoding.

The name of the clipboard stream may differ from platform to platform. On the Mac, the paste stream is "pbpaste" and the copy stream is "pbcopy". Those names are used if "darwin" is a substring of (UNIX-GETENV "ostype"). Otherwise both stream-names default to "xclip". The functions `CLIPBOARD-COPY-STREAM` and `CLIPBOARD-PASTE-STREAM` perform this selection.

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. 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 *ARRA*Y 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]

COLOR

Color is the software for driving color displays. To run COLOR, you need either a Sun (3, 4 or SPARCstations) with CG4, CG6 color hardware and color display.

The machine independent color graphics code is stored in the library files `LLCOLOR.LCOM` and `COLOR.LCOM`. The Sun color driver resides in the file `MAIKOCOLOR.LCOM` which loads `LLCOLOR.LCOM`, `COLOR.LCOM`, and `TAKEBIGBM.LCOM`.

The CG4,CG6 device supports 8 bpp (bits per pixel) at a resolution of 1152 by 900.

Note: You cannot use COLOR if you are running Medley under X.

Software Screens

To support the various hardware configurations and external display devices, the software has a special datatype, a "screen". A screen represents and controls a physical hardware display. It contains windows and icons, and tracks the mouse. There are two distinct types of screens: a black and white screen, and a color screen.

On workstations with a single hardware display, the display is shared by both the black and white screen and the color screen. It can be changed by moving the mouse cursor. The screen mode may be changed by moving the cursor out of the screen.

Turning the Color Display Software On and Off

The color display software can be turned on and off. While the color display software is on, and a small amount of processing time is used to drive the color display.

(`COLORDISPLAYP`) [Function]

Returns `T` if the color display is on; otherwise it returns `NIL`.

(`COLORDISPLAY`*ONOFF TYPE*) [Function]

Turns off the color display if *ONOFF* is set to `OFF`. If *ONOFF* is set to `ON`, it turns on the color–display–allocating memory for the color screen bitmap. *TYPE* should be `MAIKOCOLOR`, if you are using a Sun Workstation. The usual sequence of events is to `LOAD MAIKOCOLOR.LCOM` to drive GC4/GC6 color card and then to call `COLORDISPLAY` with the appropriate *TYPE* (`MAIKOCOLOR` for Sun Workstations) to turn the software on. For example,

```
(LOAD 'MAIKOCOLOR.LCOM)
(COLORDISPLAY 'ON 'MAIKOCOLOR)
```

Calling `COLORDISPLAY` allocates memory for the color screen bitmap in the sysout. Turning on the color display requires allocating the memory necessary to hold the color display screen bitmap. Turning off the color display does not free this memory; it still exists in the sysout.

Colors

The number of bits per pixel (bpp) determines the number of different colors that can be displayed at one time. When there are 8 bpp, 256 colors can be displayed at once. A table, called a *color map*, determines what color actually appears for each pixel value. A color map gives the color in terms of how much of the three primary colors (red, green, and blue) is displayed on the screen for each possible pixel value.

A color can be represented as a number, an atom, or a triple of numbers. The color map provides the final interpretation of how much red, blue, and green a color represents. A color map maps a color number ([0 . . . 2^{nbits}-1]) into the intensities of the three color guns (primary colors red, green, and blue). Each entry in the color map has 8 bits for each of the primary colors, allowing 256 levels per primary or 2^{24} possible colors (not all of which are distinct to the human eye). Within Xerox Lisp programs, colors can be manipulated as numbers, red-green-blue triples, names, or hue-lightness-saturation triples. Any function that takes a color accepts any of these different representations.

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

The following sections describe other ways to represent colors.

Red–Green–Blue Triples

A red–green–blue (RGB) triple is a list of three numbers, each of which can be between 0 and 255. The nine digits are divided as follows:

- First 3 digits = red intensity
- Second 3 digits = green intensity
- Third 3 digits = blue intensity

When an RGB triple is used, the current color map is searched to find the color with the correct intensities. If none is found, an error is generated. (That is, no attempt is made by the system to assign color numbers to intensities automatically.) An example of an RGB triple is (255 255 255), which gives the color white.

RGB [Record]

A record defined as (RED GREEN BLUE); it can be used to manipulate RGB triples.

COLORNAMES [Association list]

Maps names into colors. The CDR of the color name's entry is used as the color corresponding to the color name. This can be any of the other representations.

Note: It can even be another color name. The system does not check for loops in the name space, such as would be caused by putting '(RED . CRIMSON) and '(CRIMSON . RED) on COLORNAMES.

Some color names are available in the initial system and allow color programs written by different users to coexist. These are:

Name	**RGB**	**Number in default color maps**	
BLACK	`(0 0 0)`	15	255
BLUE	`(0 0 255)`	14	252
GREEN	`(0 255 0)`	13	227
CYAN	`(0 255 255)`	12	224
RED	`(255 0 0)`	3	31
MAGENTA	`(255 0 255)`	2	28
YELLOW	`(255 255 0)`	1	3
WHITE	`(255 255 255)`	0	0

Hue–Lightness–Saturation Triples

A hue–lightness–saturation triple is a list of three numbers:

- The first number (HUE) is an integer between 0 and 355. It indicates a position in degrees on a color wheel (blue at 0, red at 120, and green at 240).
- The second (LIGHTNESS) is a real number between zero. It indicates how much total intensity is in the color.
- The third (SATURATION) is a real number between zero and one. It indicates how disparate the three primary levels are.

`HLS` [Record]

A record defined as (HUE LIGHTNESS SATURATION); it is provided to help you manipulate HLS triples.

Example: the color blue is represented in HLS notation by `(0 .5 1.0)`.

`(COLORNUMBERP` *COLOR BITSPERPIXEL NOERRFLG*`)` [Function]

Returns the color number (offset into the screen color map) of *COLOR*. *COLOR* can be one of the following:

- A positive number less than the maximum number of colors
- A color name
- An RGB triple
- An HLS triple

If *COLOR* is one of the above and is found in the screen color map, its color number in the screen color map is returned. If not, an error is generated unless *NOERRFLG* is non-`NIL`, in which case `NIL` is returned.

`(RGBP` *X*`)` [Function]

Returns *X* if *X* is an RGB triple; otherwise it returns `NIL`.

`(HLSP` *X*`)` [Function]

Returns *X* if *X* is an HLS triple; otherwise it returns `NIL`.

Color Maps

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

(SCREENCOLORMAP *NEWCOLORMAP*) [Function]

Reads and sets the color map used by the color display. If *NEWCOLORMAP* is non-NIL, it should be a color map, and SCREENCOLORMAP sets the system color map to be that color map. The value returned is the value of the screen color map before SCREENCOLORMAP was called. If *NEWCOLORMAP* is NIL, the current screen color map is returned without change.

(CMYCOLORMAP *CYANBITS MAGENTABITS YELLOWBITS BITSPERPIXEL*) [Function]

Returns a color map that assumes the *BITSPERPIXEL* bits are to be treated as three separate color planes: *CYANBITS* bits in the cyan plane, *MAGENTABITS* bits in the magenta plane, and *YELLOWBITS* bits in the yellow plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 0 and black is 255.

(RGBCOLORMAP *REDBITS GREENBITS BLUEBITS BITSPERPIXEL*) [Function]

Returns a color map that assumes the *BITSPERPIXEL* bits are to be treated as three separate color planes: *REDBITS* bits in the red plane, *GREENBITS* bits in the green plane, and *BLUEBITS* bits in the blue plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 255 and black is 0.

(GRAYCOLORMAP *BITSPERPIXEL*) [Function]

Returns a color map containing shades of gray. White is 0 and black is 255.

(COLORMAPCREATE *INTENSITIES BITSPERPIXEL*) [Function]

Creates a color map for a screen that has *BITSPERPIXEL* bits per pixel. If *BITSPERPIXEL* is NIL, the number of bits per pixel is taken from the current color display setting. *INTENSITIES* specifies the initial colors that should be in the map. If *INTENSITIES* is not NIL, it should be a list of color specifications other than color numbers, e.g., the list of RGB triples returned by the function INTENSITIESFROMCOLOR MAP.

(INTENSITIESFROM COLORMAP *COLORMAP*) [Function]

Returns a list of the intensity levels of *COLORMAP* in a form accepted by COLORMAPCREATE. The default is SCREENCOLORMAP. This list can be written on file, providing a way of saving color map specifications.

(COLORMAPCOPY *COLORMAP BITSPERPIXEL*) [Function]

Returns a color map that contains the same color intensities as *COLORMAP* if *COLORMAP* is a color map. Otherwise, it returns a color map with default color values.

(MAPOFACOLOR *PRIMARIES*) [Function]

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

Changing Color Maps

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

(SETCOLORINTENSITY *COLORMAP COLORNUMBER COLORSPEC*) [Function]

Sets the primary intensities of color number *COLORNUMBER* in the color map *COLORMAP* to the ones specified by *COLORSPEC*. *COLORSPEC* can be either an RGB triple, an HLS triple, or a color name. The value returned is NIL.

(COLORLEVEL *COLORMAP COLORNUMBER PRIMARYNEWLEVEL*)
[Function]

Sets and reads the intensity level of the primary color *PRIMARY* (RED, GREEN, or BLUE) for the color number *COLORNUMBER* in the color map *COLORMAP*. If *NEWLEVEL* is a number between 0 and 255, it is set. The previous value of the intensity of *PRIMARY* is returned.

(ADJUSTCOLORMAP *PRIMARY DELTA COLORMAP*) [Function]

Adds *DELTA* to the intensity of the *PRIMARY* color value (RED, GREEN, or BLUE) for every color number in *COLORMAP*.

(ROTATECOLORMAP *STARTCOLOR THRUCOLOR*) [Function]

Rotates a sequence of colors in the SCREENCOLORMAP. The rotation moves the intensity values of color number *STARTCOLOR* into color number *STARTCOLOR*+1, the intensity values of color number *STARTCOLOR*+1 into color number *STARTCOLOR*+2, etc., and *THRUCOLOR's* values into *STARTCOLOR*.

(EDITCOLORMAP *VAR NOQFLG*) [Function]

Allows interactive editing of a color map. If *VAR* is an atom whose value is a color map, its value is edited. Otherwise a new color map is created and edited. The color map being edited becomes the screen color map while the editing takes place so that its effects can be observed. The edited color map is returned as the value. If *NOQFLG* is NIL and the color display is on, you are asked if you want a test pattern of colors. If you answer "yes," the function SHOWCOLORTESTPATTERN is called, which displays a test pattern showing blocks of each of the possible colors.

The system prompts for the location of a color control window to be placed on the black-and-white display. This window allows the value of any of the colors to be changed. The number of the color being edited is in the upper left part of the window. Six bars are displayed. The right three bars give the color intensities for the three primary colors of the current color number. The left three bars give the value of the color's Hue, Lightness, and Saturation parameters. To change these levels, position the mouse cursor in one of the bars and press the left mouse button. While the left button is down, the value of that parameter tracks the *Y* position of the cursor. When the left button is released, the color tracking stops. To change the color being edited, press the middle mouse button while the cursor is inside the edit window. This brings up a menu of color numbers. Select one to set the current color to the selected color.

The color being edited can also be changed by selecting the menu item PickPt. This switches the cursor onto the color screen and allows you to select a point from the color screen. It then edits the color of the selected point.

To stop the editing, move the cursor into the title of the editing window and press the middle button. This brings up a menu. Select Stop to quit.

Color Bitmaps

A color bitmap is actually a bitmap that has more than one bit per pixel. Use the function `BITSPERPIXEL` to test whether a bitmap is a color bitmap.

`(BITSPERPIXEL` *BITMAP*`)` [Function]

Returns the bits per pixel of *BITMAP*. If this does not equal one, *BITMAP* is a color bitmap.

In multiple–bit–per–pixel bitmaps, the bits that represent a pixel are stored contiguously. `BITMAPCREATE` is passed a *BITSPERPIXEL* argument to create multiple–bit–per–pixel bitmaps.

With CG4,CG6, `BITSPERPIXEL` is 8 for color bitmaps.

`(BITMAPCREATE` *WIDTH HEIGHT BITSPERPIXEL*`)` [Function]

Creates a color bitmap that is *WIDTH* pixels wide by *HEIGHT* pixels high allowing *BITSPERPIXEL* bits per pixel. Currently any value of *BITSPERPIXEL* except `1`, `4`, `8`, or `NIL` (defaulting to `1`) causes an error.

`BITMAPCREATE` may return two types of bitmap. An ordinary `BITMAP` type usually contains up to 131066 cells (32 bits of data) in it. A larger bitmap can be created as `BIGBM`.

```
(DATATYPE BIGBM (BIGBMWIDTH BIGBMHEIGHT BIGBMLIST)
```

`BIGBM` has the list of pointers which point to `BITMAP`s.

An 8 bpp screen bitmap (1052 * 900) uses approximately 1 Mbyte.

`(COLORSCREENBITMAP)` [Function]

Returns the bitmap that is being or will be displayed on the color display. This is `NIL` if the color display has never been turned on (see `COLORDISPLAY` below).

Screens, Screenpositions, and Screenregions

In addition to positions and regions, you need to be aware of screens, screenpositions, and screenregions in the presence of multiple screens.

Screens

SCREEN [Datatype]

There are generally two screen datatype instances in existence when working with color. This is because you are attached to two displays, a black and white display and a color display.

(MAINSCREEN) [Function]

Returns the screen datatype instance that represents the black and white screen. This will be something like {SCREEN}#74,24740.

(COLORSCREEN) [Function]

Returns the screen datatype instance that represents the color screen. Screens appear as part of screenpositions and screenregions, serving as the extra information needed to make clear whether a particular position or region should be viewed as lying on the black and white display or the color display.

(SCREENBITMAP *SCREEN*) [Function]

Returns the bitmap destination of *SCREEN*. If *SCREEN* is NIL, it returns the black and white screen bitmap.

Screenpositions

SCREENPOSITION [Record]

Somewhat like a position, a screenposition denotes a point in an X,Y coordinate system on a particular screen. Screenpositions are defined according to the following record declaration:

```
(RECORD SCREENPOSITION (SCREEN . POSITION)
     (SUBRECORD POSITION))
```

A SCREENPOSITION is an instance of a record with fields XCOORD, YCOORD, and SCREEN and is manipulated with the standard record package facilities. For example, (create SCREENPOSITION XCOORD _ 10 YCOORD _ 20 SCREEN _ (COLORSCREEN)) creates a screenposition representing the point (10,20) on the color display. You can extract the position of a screenposition by fetching its POSITION. For example, (fetch (SCREENPOSITION POSITION) of SP12).

Screenregions

SCREENREGION [Record]

Similar to a region, a screenregion denotes a rectangular area in a coordinate system. Screenregions are defined according to the following record declaration:

```
(RECORD SCREENREGION (SCREEN . REGION)(SUBRECORD REGION))
```

A screenregion is characterized by the coordinates of its bottom left corner and its width and height. A SCREENREGION is a record with fields LEFT, BOTTOM, WIDTH, HEIGHT, and SCREEN. It can be manipulated with the standard record package facilities. There are access functions for the REGION record that return the TOP and RIGHT of the region. You can extract the region of a screenregion by fetching its REGION. For example, (fetch (SCREENREGION REGION) of SR8).

Screenposition and Screenregion Prompting

The following functions can be used by programs to allow you to interactively specify screenpositions or screenregions on a display screen.

(GETSCREENPOSITION *WINDOW CURSOR*) [Function]

Similar to GETPOSITION. Returns a SCREENPOSITION you specifie. GETSCREENPOSITION waits for you to press and release the left button of the mouse and returns the cursor screenposition at the time of release. If *WINDOW* is a WINDOW, the screenposition is on the same screen as *WINDOW* and in the coordinate system of *WINDOW*'s display stream. If *WINDOW* is NIL, the position is in screen coordinates.

(GETBOXSCREENPOSITION *BOXWIDTH BOXHEIGHT ORGX ORGY WINDOW PROMPTMSG*) [Function]

Similar to GETBOXPOSITION. Returns a SCREENPOSITION you specified. This function allows you to position a "ghost" region of size *BOXWIDTH* by *BOXHEIGHT* on a screen, and returns the SCREENPOSITION of the lower left corner of the screenregion chosen. A ghost region is locked to the cursor so that if the cursor is moved, the ghost region moves with it. To change to another corner, press and hold the right button. With the right button down, the cursor can be moved across a screen or to other screens without effect on the ghost region frame. When the right button is released, the mouse snaps to the nearest corner, which then becomes locked to the cursor. (To change the held corner after the left or middle button is down, hold both the original button and the right button down while moving the cursor to the desired new corner, then release just the right button.) When the left or middle button is pressed and released, the lower left corner of the screenregion chosen at the time of release is returned. If *WINDOW* is a WINDOW, the screenposition is on the same screen as *WINDOW* and in the coordinate system of *WINDOW*'s display stream. If *WINDOW* is NIL, the position is in the screen.

(GETSCREENREGION *MINWIDTH MINHEIGHT OLDREGION NEWREGIONFN NEWREGIONFNARG INITCORNERS*) [Function]

Similar to GETREGION. Returns a SCREENREGION you specified. This function allows you to specify a new screenregion and returns that screenregion. GETSCREENREGION prompts for a screenregion by displaying a four-pronged box next to the cursor arrow at one corner of a "ghost" region: . If you press the left button, the corner of a "ghost" screenregion opposite the cursor is locked where it is. Once one corner has been fixed, the ghost screenregion expands as the cursor moves.

To specify a screenregion:

1. Move the ghost box so that the corner opposite the cursor is at one corner of the intended screenregion.
2. Press the left button.
3. Move the cursor to the screenposition of the opposite corner of the intended screenregion while holding down the left button
4. Release the left button.

Before one corner has been fixed, you can switch the cursor to another corner of the ghost screenregion by holding down the right button. With the right button down, the cursor changes to a "forceps" () and the cursor can be moved across a screen or to other screens without effect on the ghost screenregion frame. When the right button is released, the cursor snaps to the nearest corner of the ghost screenregion.

After one corner has been fixed, you can still switch to another corner. To change to another corner, continue to hold down the left button and hold down the right button also. With both buttons down, the cursor can be moved across a screen or to other screens without effect on the ghost screenregion frame. When the right button is released, the cursor snaps to the nearest corner, which becomes the moving corner. In this way, the screenregion may be moved all over a screen, and to other screens, before its size and screenposition are finalized.

The size of the initial ghost screenregion is controlled by the *MINWIDTH*, *MINHEIGHT*, *OLDREGION*, and *INITCORNERS* arguments.

(GETBOXSCREENREGION *WIDTH HEIGHT ORGX ORGY WINDOW PROMPTMSG*) [Function]

Similar to GETBOXREGION. Returns a SCREENREGION you specified. This function performs the same prompting as GETBOXSCREENPOSITION and returns the SCREENREGION specified instead of the SCREENPOSITION of its lower left corner.

Color Windows and Menus

The Medley window system provides both interactive and programmatic constructs for creating, moving, reshaping, overlapping, and destroying windows in such a way that a program can use a window in a relatively transparent fashion . Menus are a special type of window provided by the window system, used for displaying a set of items, on which you use the mouse cursor to make a selection. The menu facility also allows you to create and use menus in interactive programs.

(CREATEW *REGION TITLE BORDERSIZE NOOPENFLG*) [Function]

Creates a new window. *REGION* indicates where and how large the window should be by specifying the exterior screenregion of the window. In a user environment with multiple screens, such as a black and white screen and color screen both connected to the same machine, there is a new special problem in indicating which screen the *REGION* is supposed to be a region of. To resolve this problem, allow CREATEW to take screenregion arguments as *REGION*.

For example:

```
(SETQ FOO (CREATEW (CREATE SCREENREGION
    SCREEN _ (COLORSCREEN)
    LEFT _ 20  BOTTOM _ 210
    WIDTH _ 290 HEIGHT _ 170)
    "FOO WINDOW"))
```

creates a window titled "FOO WINDOW" on the color screen. To create a window on the black and white screen, use SCREEN _ (MAINSCREEN) in the CREATE SCREENREGION expression.

Note: It is still perfectly legal to pass in a *REGION* that is a region, not a screenregion, to CREATEW, but it is better to pass screenregions. When *REGION* is a region, *REGION* is disambiguated in a somewhat arbitrary manner that may not always turn out to be what you want.

When *REGION* is a region, *REGION* is disambiguated by coercing *REGION* to be a screenregion on the screen which currently contains the cursor. Software calling CREATEW with regions instead of screenregions tends to do what you expect.

(WINDOWPROP *WINDOW PROP NEWVALUE*) [NoSpread Function]

If *PROP* is 'SCREEN, then WINDOWPROP returns the screen *WINDOW* is on. If *NEWVALUE* is given (even if given as NIL), with *PROP* is 'SCREEN, then WINDOWPROP generates an error. Any other *PROP* name is handled in the usual way.

(OPENWINDOWS *SCREEN*) [Function]

Returns a list of all open windows on *SCREEN* if *SCREEN* is a screen datatype such as (MAINSCREEN) or (COLORSCREEN). If *SCREEN* is NIL, then *SCREEN* defaults to the screen containing the cursor. If *SCREEN* is T, a list of all open windows on all screens is returned.

Fonts in Color

There is no special function for creating color fonts.You use the same font on every screen.

You can use color characters by specifying the foreground color (the color of the characters of the font) and background color (the color of the background behind the characters that gets printed along with the characters) in the DISPLAYSTREAM data used for the current window (see below).

Using Color

The current color implementation allows display streams to operate on color bitmaps. The two functions DSPCOLOR and DSPBACKCOLOR set the color in which a stream draws when the user defaults a color argument to a drawing function and printing characters.

(DSPCOLOR *COLOR STREAM*) [Function]

Sets the foreground color of a stream. It returns the previous foreground color. If *COLOR* is NIL, it returns the current foreground color without changing anything. The default foreground color is MINIMUMCOLOR=0, which is white in the default color maps.

(DSPBACKCOLOR *COLOR STREAM*) [Function]

Sets the background color of a stream. It returns the previous background color. If *COLOR* is NIL, it returns the current background color without changing anything. The default background color is (MAXIMUMCOLOR BITSPERPIXEL)=255, which is black in the default color maps.

The BITBLT, line–drawing and curve–drawing routines know how to operate on a color-capable stream. Following are some notes about them.

BITBLTing in Color

If you are BITBLTing from a color bitmap onto another color bitmap of the same bpp, the operations PAINT, INVERT, and ERASE are done on a bit level, not on a pixel level. Thus, painting color 3 onto color 10 results in color 11.

If BITBLTing from a black–and–white bitmap onto a color bitmap, the 1 bits appear in the DSPCOLOR, and the 0 bits in DSPBACKCOLOR. BLTing from black–and–white to color is fairly expensive; if the same bitmap is going to be put up several times in the same color, it is faster to create a color copy and then to BLT the color copy.

If the source type is TEXTURE and the destination bitmap is a color bitmap, the *Texture* argument is taken to be a color. Therefore, to fill an area with the color BLUE assuming COLORSTR is a stream whose destination is the color screen, use `(BITBLT NIL NIL NIL COLORSTR 50 75 100 200 'TEXTURE 'REPLACE 'BLUE)`.

Drawing Curves and Lines in Color

For the functions DRAWCIRCLE, DRAWELLIPSE, and DRAWCURVE, the notion of a brush has been extended to include a color. A BRUSH is now (BRUSHSHAPE BRUSHSIZE BRUSHCOLOR). Also, a brush can be a bitmap (which can be a color bitmap).

Line-drawing routines take a color argument which is the color of the line if the destination of the display stream is a color bitmap.

(DRAWLINE *X1 Y1 X2 Y2 WIDTH OPERATION STREAM COLOR*) [Function]

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

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

(DRAWBETWEEN *POS1 POS2 WIDTH OPERATION STREAM COLOR*) [Function]

If the *COLOR* argument is NIL, the DSPCOLOR of the stream is used.

Printing in Color

Printing only works in REPLACE mode. The characters have a background color and a foreground color determined by the DSPCOLOR and DSPBACKCOLOR .

Example of printing to an 8 bpp color screen:

```
(SETQ FOO (CREATEW (CREATE SCREENREGION
      SCREEN _ (COLORSCREEN)   LEFT _ 20
      BOTTOM _ 210     WIDTH _ 290
      HEIGHT _ 170)    "FOO WINDOW"))

(DSPCOLOR FOO 'GREEN)

(DSPBACKCOLOR 'YELLOW)
```

`(PRINT 'HELLO FOO)` prints in green against a yellow background.

Operating the Cursor on the Color Screen

To move the cursor to the color screen, slide the cursor off the left or right edge of the black and white screen onto the color screen, or call the `CURSORPOSITION` or `CURSORSCREEN` function.

`(WORPCURSOR` *ENABLE*`)` [Function]

If *ENABLE* is NIL, you cannot exit the current screen by sliding the cursor off.

`(CURSORPOSITION` *NEWPOSITION* `--)` [Function]

NEWPOSITION can be a position or a screenposition.

`(CURSORSCREEN` *SCREEN XCOORD YCOORD*`)` [Function]

Moves the cursor to the screenposition determined by *SCREEN*, *XCOORD*, and *YCOORD*. *SCREEN* should be the value of either `(COLORSCREEN)` or `(MAINSCREEN)`.

While on the color screen, the cursor is placed by doing `BITBLT`s in software rather than with microcode and hardware as with the black and white cursor. It is automatically taken down whenever an operation is performed that changes any bits on the color screen. The speed of the color cursor compares well with that of the black–and–white cursor but there can be a noticeable flicker when there is much input/output to the color screen. While the cursor is on the color screen, the black-and-white cursor is cleared giving the appearance that there is never more than one cursor at a given time.

Miscellaneous Color Functions

`(COLORIZEBITMAP` *BITMAP 0COLOR 1COLOR BITSPERPIXEL*`)` [Function]

Creates a color bitmap from a black–and–white bitmap. The returned bitmap has color number *1COLOR* in those pixels of *BITMAP* that were 1 and *0COLOR* in those pixels of *BITMAP* that were 0. This provides a way of producing a color bitmap from a black–and –white bitmap.

`(UNCOLORIZEBITMAP` *BITMAP COLORMAP*`)` [Function]

Creates a black–and–white bitmap from a color bitmap.

`(SHOWCOLORTESTPATTERN` *BARSIZE*`)` [Function]

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

[This page intentionally left blank]

COLOR

Color is the software for driving color displays. To run COLOR, you need either a Sun (3, 4 or SPARCstations) with CG4, CG6 color hardware and color display.

The machine independent color graphics code is stored in the library files LLCOLOR.LCOM and COLOR.LCOM. The Sun color driver resides in the file MAIKOCOLOR.LCOM which loads LLCOLOR.LCOM, COLOR.LCOM, and TAKEBIGBM.LCOM.

The CG4,CG6 device supports 8 bpp (bits per pixel) at a resolution of 1152 by 900.

Note: You cannot use COLOR if you are running Medley under X.

Software Screens

To support the various hardware configurations and external display devices, the software has a special datatype, a "screen". A screen represents and controls a physical hardware display. It contains windows and icons, and tracks the mouse. There are two distinct types of screens: a black and white screen, and a color screen.

On workstations with a single hardware display, the display is shared by both the black and white screen and the color screen. It can be changed by moving the mouse cursor. The screen mode may be changed by moving the cursor out of the screen.

Turning the Color Display Software On and Off

The color display software can be turned on and off. While the color display software is on, and a small amount of processing time is used to drive the color display.

(COLORDISPLAYP) [Function]

Returns T if the color display is on; otherwise it returns NIL.

(COLORDISPLAY*ONOFF TYPE*) [Function]

Turns off the color display if *ONOFF* is set to OFF. If *ONOFF* is set to ON, it turns on the color–display–allocating memory for the color screen bitmap. *TYPE* should be MAIKOCOLOR, if you are using a Sun Workstation. The usual sequence of events is to LOAD MAIKOCOLOR.LCOM to drive GC4/GC6 color card and then to call COLORDISPLAY with the appropriate *TYPE* (MAIKOCOLOR for Sun Workstations) to turn the software on. For example,

```
(LOAD 'MAIKOCOLOR.LCOM)
(COLORDISPLAY 'ON 'MAIKOCOLOR)
```

Calling COLORDISPLAY allocates memory for the color screen bitmap in the sysout. Turning on the color display requires allocating the memory necessary to hold the color display screen bitmap. Turning off the color display does not free this memory; it still exists in the sysout.

Colors

The number of bits per pixel (bpp) determines the number of different colors that can be displayed at one time. When there are 8 bpp, 256 colors can be displayed at once. A table, called a *color map*, determines what color actually appears for each pixel value. A color map gives the color in terms of how much of the three primary colors (red, green, and blue) is displayed on the screen for each possible pixel value.

A color can be represented as a number, an atom, or a triple of numbers. The color map provides the final interpretation of how much red, blue, and green a color represents. A color map maps a color number ([0 . . . 2^{nbits}-1]) into the intensities of the three color guns (primary colors red, green, and blue). Each entry in the color map has 8 bits for each of the primary colors, allowing 256 levels per primary or 2^{24} possible colors (not all of which are distinct to the human eye). Within Xerox Lisp programs, colors can be manipulated as numbers, red-green-blue triples, names, or hue-lightness-saturation triples. Any function that takes a color accepts any of these different representations.

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

The following sections describe other ways to represent colors.

Red–Green–Blue Triples

A red–green–blue (RGB) triple is a list of three numbers, each of which can be between 0 and 255. The nine digits are divided as follows:

- First 3 digits = red intensity
- Second 3 digits = green intensity
- Third 3 digits = blue intensity

When an RGB triple is used, the current color map is searched to find the color with the correct intensities. If none is found, an error is generated. (That is, no attempt is made by the system to assign color numbers to intensities automatically.) An example of an RGB triple is (255 255 255), which gives the color white.

`RGB` [Record]

A record defined as (`RED GREEN BLUE`); it can be used to manipulate RGB triples.

`COLORNAMES` [Association list]

Maps names into colors. The CDR of the color name's entry is used as the color corresponding to the color name. This can be any of the other representations.

Note: It can even be another color name. The system does not check for loops in the name space, such as would be caused by putting `'(RED . CRIMSON)` and `'(CRIMSON . RED)` on `COLORNAMES`.

Some color names are available in the initial system and allow color programs written by different users to coexist. These are:

Name	**RGB**	**Number in default color maps**	
BLACK	`(0 0 0)`	15	255
BLUE	`(0 0 255)`	14	252
GREEN	`(0 255 0)`	13	227
CYAN	`(0 255 255)`	12	224
RED	`(255 0 0)`	3	31
MAGENTA	`(255 0 255)`	2	28
YELLOW	`(255 255 0)`	1	3
WHITE	`(255 255 255)`	0	0

Hue–Lightness–Saturation Triples

A hue–lightness–saturation triple is a list of three numbers:

- The first number (HUE) is an integer between 0 and 355. It indicates a position in degrees on a color wheel (blue at 0, red at 120, and green at 240).
- The second (LIGHTNESS) is a real number between zero. It indicates how much total intensity is in the color.
- The third (SATURATION) is a real number between zero and one. It indicates how disparate the three primary levels are.

`HLS` [Record]

A record defined as (HUE LIGHTNESS SATURATION); it is provided to help you manipulate HLS triples.

Example: the color blue is represented in HLS notation by `(0 .5 1.0)`.

`(COLORNUMBERP` *COLOR BITSPERPIXEL NOERRFLG*`)` [Function]

Returns the color number (offset into the screen color map) of *COLOR*. *COLOR* can be one of the following:

- A positive number less than the maximum number of colors
- A color name
- An RGB triple
- An HLS triple

If *COLOR* is one of the above and is found in the screen color map, its color number in the screen color map is returned. If not, an error is generated unless *NOERRFLG* is non-`NIL`, in which case `NIL` is returned.

`(RGBP` *X*`)` [Function]

Returns *X* if *X* is an RGB triple; otherwise it returns `NIL`.

`(HLSP` *X*`)` [Function]

Returns *X* if *X* is an HLS triple; otherwise it returns `NIL`.

Color Maps

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

(SCREENCOLORMAP *NEWCOLORMAP*) [Function]

Reads and sets the color map used by the color display. If *NEWCOLORMAP* is non-NIL, it should be a color map, and SCREENCOLORMAP sets the system color map to be that color map. The value returned is the value of the screen color map before SCREENCOLORMAP was called. If *NEWCOLORMAP* is NIL, the current screen color map is returned without change.

(CMYCOLORMAP *CYANBITS MAGENTABITS YELLOWBITS BITSPERPIXEL*) [Function]

Returns a color map that assumes the *BITSPERPIXEL* bits are to be treated as three separate color planes: *CYANBITS* bits in the cyan plane, *MAGENTABITS* bits in the magenta plane, and *YELLOWBITS* bits in the yellow plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 0 and black is 255.

(RGBCOLORMAP *REDBITS GREENBITS BLUEBITS BITSPERPIXEL*) [Function]

Returns a color map that assumes the *BITSPERPIXEL* bits are to be treated as three separate color planes: *REDBITS* bits in the red plane, *GREENBITS* bits in the green plane, and *BLUEBITS* bits in the blue plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 255 and black is 0.

(GRAYCOLORMAP *BITSPERPIXEL*) [Function]

Returns a color map containing shades of gray. White is 0 and black is 255.

(COLORMAPCREATE *INTENSITIES BITSPERPIXEL*) [Function]

Creates a color map for a screen that has *BITSPERPIXEL* bits per pixel. If *BITSPERPIXEL* is NIL, the number of bits per pixel is taken from the current color display setting. *INTENSITIES* specifies the initial colors that should be in the map. If *INTENSITIES* is not NIL, it should be a list of color specifications other than color numbers, e.g., the list of RGB triples returned by the function INTENSITIESFROMCOLOR MAP.

(INTENSITIESFROM COLORMAP *COLORMAP*) [Function]

Returns a list of the intensity levels of *COLORMAP* in a form accepted by COLORMAPCREATE. The default is SCREENCOLORMAP. This list can be written on file, providing a way of saving color map specifications.

(COLORMAPCOPY *COLORMAP BITSPERPIXEL*) [Function]

Returns a color map that contains the same color intensities as *COLORMAP* if *COLORMAP* is a color map. Otherwise, it returns a color map with default color values.

(MAPOFACOLOR *PRIMARIES*) [Function]

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

Changing Color Maps

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

(SETCOLORINTENSITY *COLORMAP COLORNUMBER COLORSPEC*) [Function]

Sets the primary intensities of color number *COLORNUMBER* in the color map *COLORMAP* to the ones specified by *COLORSPEC*. *COLORSPEC* can be either an RGB triple, an HLS triple, or a color name. The value returned is NIL.

(COLORLEVEL *COLORMAP COLORNUMBER PRIMARYNEWLEVEL*) [Function]

Sets and reads the intensity level of the primary color *PRIMARY* (RED, GREEN, or BLUE) for the color number *COLORNUMBER* in the color map *COLORMAP*. If *NEWLEVEL* is a number between 0 and 255, it is set. The previous value of the intensity of *PRIMARY* is returned.

(ADJUSTCOLORMAP *PRIMARY DELTA COLORMAP*) [Function]

Adds *DELTA* to the intensity of the *PRIMARY* color value (RED, GREEN, or BLUE) for every color number in *COLORMAP*.

(ROTATECOLORMAP *STARTCOLOR THRUCOLOR*) [Function]

Rotates a sequence of colors in the SCREENCOLORMAP. The rotation moves the intensity values of color number *STARTCOLOR* into color number *STARTCOLOR*+1, the intensity values of color number *STARTCOLOR*+1 into color number *STARTCOLOR*+2, etc., and *THRUCOLOR's* values into *STARTCOLOR*.

(EDITCOLORMAP *VAR NOQFLG*) [Function]

Allows interactive editing of a color map. If *VAR* is an atom whose value is a color map, its value is edited. Otherwise a new color map is created and edited. The color map being edited becomes the screen color map while the editing takes place so that its effects can be observed. The edited color map is returned as the value. If *NOQFLG* is NIL and the color display is on, you are asked if you want a test pattern of colors. If you answer "yes," the function SHOWCOLORTESTPATTERN is called, which displays a test pattern showing blocks of each of the possible colors.

The system prompts for the location of a color control window to be placed on the black-and-white display. This window allows the value of any of the colors to be changed. The number of the color being edited is in the upper left part of the window. Six bars are displayed. The right three bars give the color intensities for the three primary colors of the current color number. The left three bars give the value of the color's Hue, Lightness, and Saturation parameters. To change these levels, position the mouse cursor in one of the bars and press the left mouse button. While the left button is down, the value of that parameter tracks the *Y* position of the cursor. When the left button is released, the color tracking stops. To change the color being edited, press the middle mouse button while the cursor is inside the edit window. This brings up a menu of color numbers. Select one to set the current color to the selected color.

The color being edited can also be changed by selecting the menu item PickPt. This switches the cursor onto the color screen and allows you to select a point from the color screen. It then edits the color of the selected point.

To stop the editing, move the cursor into the title of the editing window and press the middle button. This brings up a menu. Select Stop to quit.

Color Bitmaps

A color bitmap is actually a bitmap that has more than one bit per pixel. Use the function `BITSPERPIXEL` to test whether a bitmap is a color bitmap.

`(BITSPERPIXEL` *BITMAP*`)` [Function]

Returns the bits per pixel of *BITMAP*. If this does not equal one, *BITMAP* is a color bitmap.

In multiple–bit–per–pixel bitmaps, the bits that represent a pixel are stored contiguously. `BITMAPCREATE` is passed a *BITSPERPIXEL* argument to create multiple–bit–per–pixel bitmaps.

With CG4,CG6, `BITSPERPIXEL` is 8 for color bitmaps.

`(BITMAPCREATE` *WIDTH HEIGHT BITSPERPIXEL*`)` [Function]

Creates a color bitmap that is *WIDTH* pixels wide by *HEIGHT* pixels high allowing *BITSPERPIXEL* bits per pixel. Currently any value of *BITSPERPIXEL* except `1`, `4`, `8`, or `NIL` (defaulting to `1`) causes an error.

`BITMAPCREATE` may return two types of bitmap. An ordinary `BITMAP` type usually contains up to 131066 cells (32 bits of data) in it. A larger bitmap can be created as `BIGBM`.

```
(DATATYPE BIGBM (BIGBMWIDTH BIGBMHEIGHT BIGBMLIST)
```

`BIGBM` has the list of pointers which point to `BITMAP`s.

An 8 bpp screen bitmap (1052 * 900) uses approximately 1 Mbyte.

`(COLORSCREENBITMAP)` [Function]

Returns the bitmap that is being or will be displayed on the color display. This is `NIL` if the color display has never been turned on (see `COLORDISPLAY` below).

Screens, Screenpositions, and Screenregions

In addition to positions and regions, you need to be aware of screens, screenpositions, and screenregions in the presence of multiple screens.

Screens

`SCREEN` [Datatype]

There are generally two screen datatype instances in existence when working with color. This is because you are attached to two displays, a black and white display and a color display.

`(MAINSCREEN)` [Function]

Returns the screen datatype instance that represents the black and white screen. This will be something like `{SCREEN}#74,24740`.

`(COLORSCREEN)` [Function]

Returns the screen datatype instance that represents the color screen. Screens appear as part of screenpositions and screenregions, serving as the extra information needed to make clear whether a particular position or region should be viewed as lying on the black and white display or the color display.

`(SCREENBITMAP` *SCREEN*`)` [Function]

Returns the bitmap destination of *SCREEN*. If *SCREEN* is `NIL`, it returns the black and white screen bitmap.

Screenpositions

`SCREENPOSITION` [Record]

Somewhat like a position, a screenposition denotes a point in an X,Y coordinate system on a particular screen. Screenpositions are defined according to the following record declaration:

```
(RECORD SCREENPOSITION (SCREEN . POSITION)
     (SUBRECORD POSITION))
```

A `SCREENPOSITION` is an instance of a record with fields `XCOORD`, `YCOORD`, and `SCREEN` and is manipulated with the standard record package facilities. For example, `(create SCREENPOSITION XCOORD _ 10 YCOORD _ 20 SCREEN _ (COLORSCREEN))` creates a screenposition representing the point (10,20) on the color display. You can extract the position of a screenposition by fetching its `POSITION`. For example, `(fetch (SCREENPOSITION POSITION) of SP12)`.

Screenregions

`SCREENREGION` [Record]

Similar to a region, a screenregion denotes a rectangular area in a coordinate system. Screenregions are defined according to the following record declaration:

```
(RECORD SCREENREGION (SCREEN . REGION)(SUBRECORD REGION))
```

A screenregion is characterized by the coordinates of its bottom left corner and its width and height. A `SCREENREGION` is a record with fields `LEFT`, `BOTTOM`, `WIDTH`, `HEIGHT`, and `SCREEN`. It can be manipulated with the standard record package facilities. There are access functions for the `REGION` record that return the `TOP` and `RIGHT` of the region. You can extract the region of a screenregion by fetching its `REGION`. For example, `(fetch (SCREENREGION REGION) of SR8)`.

Screenposition and Screenregion Prompting

The following functions can be used by programs to allow you to interactively specify screenpositions or screenregions on a display screen.

(GETSCREENPOSITION *WINDOW CURSOR*) [Function]

Similar to GETPOSITION. Returns a SCREENPOSITION you specifie. GETSCREENPOSITION waits for you to press and release the left button of the mouse and returns the cursor screenposition at the time of release. If *WINDOW* is a WINDOW, the screenposition is on the same screen as *WINDOW* and in the coordinate system of *WINDOW*'s display stream. If *WINDOW* is NIL, the position is in screen coordinates.

(GETBOXSCREENPOSITION *BOXWIDTH BOXHEIGHT ORGX ORGY WINDOW PROMPTMSG*) [Function]

Similar to GETBOXPOSITION. Returns a SCREENPOSITION you specified. This function allows you to position a "ghost" region of size *BOXWIDTH* by *BOXHEIGHT* on a screen, and returns the SCREENPOSITION of the lower left corner of the screenregion chosen. A ghost region is locked to the cursor so that if the cursor is moved, the ghost region moves with it. To change to another corner, press and hold the right button. With the right button down, the cursor can be moved across a screen or to other screens without effect on the ghost region frame. When the right button is released, the mouse snaps to the nearest corner, which then becomes locked to the cursor. (To change the held corner after the left or middle button is down, hold both the original button and the right button down while moving the cursor to the desired new corner, then release just the right button.) When the left or middle button is pressed and released, the lower left corner of the screenregion chosen at the time of release is returned. If *WINDOW* is a WINDOW, the screenposition is on the same screen as *WINDOW* and in the coordinate system of *WINDOW*'s display stream. If *WINDOW* is NIL, the position is in the screen.

(GETSCREENREGION *MINWIDTH MINHEIGHT OLDREGION NEWREGIONFN NEWREGIONFNARG INITCORNERS*) [Function]

Similar to GETREGION. Returns a SCREENREGION you specified. This function allows you to specify a new screenregion and returns that screenregion. GETSCREENREGION prompts for a screenregion by displaying a four-pronged box next to the cursor arrow at one corner of a "ghost" region: . If you press the left button, the corner of a "ghost" screenregion opposite the cursor is locked where it is. Once one corner has been fixed, the ghost screenregion expands as the cursor moves.

To specify a screenregion:

1. Move the ghost box so that the corner opposite the cursor is at one corner of the intended screenregion.
2. Press the left button.
3. Move the cursor to the screenposition of the opposite corner of the intended screenregion while holding down the left button
4. Release the left button.

Before one corner has been fixed, you can switch the cursor to another corner of the ghost screenregion by holding down the right button. With the right button down, the cursor changes to a "forceps" () and the cursor can be moved across a screen or to other screens without effect on the ghost screenregion frame. When the right button is released, the cursor snaps to the nearest corner of the ghost screenregion.

After one corner has been fixed, you can still switch to another corner. To change to another corner, continue to hold down the left button and hold down the right button also. With both buttons down, the cursor can be moved across a screen or to other screens without effect on the ghost screenregion frame. When the right button is released, the cursor snaps to the nearest corner, which becomes the moving corner. In this way, the screenregion may be moved all over a screen, and to other screens, before its size and screenposition are finalized.

The size of the initial ghost screenregion is controlled by the *MINWIDTH*, *MINHEIGHT*, *OLDREGION*, and *INITCORNERS* arguments.

(GETBOXSCREENREGION *WIDTH HEIGHT ORGX ORGY WINDOW PROMPTMSG*) [Function]

Similar to GETBOXREGION. Returns a SCREENREGION you specified. This function performs the same prompting as GETBOXSCREENPOSITION and returns the SCREENREGION specified instead of the SCREENPOSITION of its lower left corner.

Color Windows and Menus

The Medley window system provides both interactive and programmatic constructs for creating, moving, reshaping, overlapping, and destroying windows in such a way that a program can use a window in a relatively transparent fashion . Menus are a special type of window provided by the window system, used for displaying a set of items, on which you use the mouse cursor to make a selection. The menu facility also allows you to create and use menus in interactive programs.

(CREATEW *REGION TITLE BORDERSIZE NOOPENFLG*) [Function]

Creates a new window. *REGION* indicates where and how large the window should be by specifying the exterior screenregion of the window. In a user environment with multiple screens, such as a black and white screen and color screen both connected to the same machine, there is a new special problem in indicating which screen the *REGION* is supposed to be a region of. To resolve this problem, allow CREATEW to take screenregion arguments as *REGION*.

For example:

```
(SETQ FOO (CREATEW (CREATE SCREENREGION
    SCREEN _ (COLORSCREEN)
    LEFT _ 20  BOTTOM _ 210
    WIDTH _ 290 HEIGHT _ 170)
    "FOO WINDOW"))
```

creates a window titled "FOO WINDOW" on the color screen. To create a window on the black and white screen, use SCREEN _ (MAINSCREEN) in the CREATE SCREENREGION expression.

Note: It is still perfectly legal to pass in a *REGION* that is a region, not a screenregion, to CREATEW, but it is better to pass screenregions. When *REGION* is a region, *REGION* is disambiguated in a somewhat arbitrary manner that may not always turn out to be what you want.

When *REGION* is a region, *REGION* is disambiguated by coercing *REGION* to be a screenregion on the screen which currently contains the cursor. Software calling CREATEW with regions instead of screenregions tends to do what you expect.

(WINDOWPROP *WINDOW PROP NEWVALUE*) [NoSpread Function]

If *PROP* is 'SCREEN, then WINDOWPROP returns the screen *WINDOW* is on. If *NEWVALUE* is given (even if given as NIL), with *PROP* is 'SCREEN, then WINDOWPROP generates an error. Any other *PROP* name is handled in the usual way.

(OPENWINDOWS *SCREEN*) [Function]

Returns a list of all open windows on *SCREEN* if *SCREEN* is a screen datatype such as (MAINSCREEN) or (COLORSCREEN). If *SCREEN* is NIL, then *SCREEN* defaults to the screen containing the cursor. If *SCREEN* is T, a list of all open windows on all screens is returned.

Fonts in Color

There is no special function for creating color fonts.You use the same font on every screen.

You can use color characters by specifying the foreground color (the color of the characters of the font) and background color (the color of the background behind the characters that gets printed along with the characters) in the DISPLAYSTREAM data used for the current window (see below).

Using Color

The current color implementation allows display streams to operate on color bitmaps. The two functions DSPCOLOR and DSPBACKCOLOR set the color in which a stream draws when the user defaults a color argument to a drawing function and printing characters.

(DSPCOLOR *COLOR STREAM*) [Function]

Sets the foreground color of a stream. It returns the previous foreground color. If *COLOR* is NIL, it returns the current foreground color without changing anything. The default foreground color is MINIMUMCOLOR=0, which is white in the default color maps.

(DSPBACKCOLOR *COLOR STREAM*) [Function]

Sets the background color of a stream. It returns the previous background color. If *COLOR* is NIL, it returns the current background color without changing anything. The default background color is (MAXIMUMCOLOR BITSPERPIXEL)=255, which is black in the default color maps.

The BITBLT, line–drawing and curve–drawing routines know how to operate on a color-capable stream. Following are some notes about them.

BITBLTing in Color

If you are BITBLTing from a color bitmap onto another color bitmap of the same bpp, the operations PAINT, INVERT, and ERASE are done on a bit level, not on a pixel level. Thus, painting color 3 onto color 10 results in color 11.

If BITBLTing from a black–and–white bitmap onto a color bitmap, the 1 bits appear in the DSPCOLOR, and the 0 bits in DSPBACKCOLOR. BLTing from black–and–white to color is fairly expensive; if the same bitmap is going to be put up several times in the same color, it is faster to create a color copy and then to BLT the color copy.

If the source type is TEXTURE and the destination bitmap is a color bitmap, the *Texture* argument is taken to be a color. Therefore, to fill an area with the color BLUE assuming COLORSTR is a stream whose destination is the color screen, use (BITBLT NIL NIL NIL COLORSTR 50 75 100 200 'TEXTURE 'REPLACE 'BLUE).

Drawing Curves and Lines in Color

For the functions DRAWCIRCLE, DRAWELLIPSE, and DRAWCURVE, the notion of a brush has been extended to include a color. A BRUSH is now (BRUSHSHAPE BRUSHSIZE BRUSHCOLOR). Also, a brush can be a bitmap (which can be a color bitmap).

Line-drawing routines take a color argument which is the color of the line if the destination of the display stream is a color bitmap.

(DRAWLINE *X1 Y1 X2 Y2 WIDTH OPERATION STREAM COLOR*) [Function]

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

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

(DRAWBETWEEN *POS1 POS2 WIDTH OPERATION STREAM COLOR*) [Function]

If the *COLOR* argument is NIL, the DSPCOLOR of the stream is used.

Printing in Color

Printing only works in REPLACE mode. The characters have a background color and a foreground color determined by the DSPCOLOR and DSPBACKCOLOR .

Example of printing to an 8 bpp color screen:

```
(SETQ FOO (CREATEW (CREATE SCREENREGION
      SCREEN _ (COLORSCREEN)   LEFT _ 20
      BOTTOM _ 210     WIDTH _ 290
      HEIGHT _ 170)    "FOO WINDOW"))

(DSPCOLOR FOO 'GREEN)

(DSPBACKCOLOR 'YELLOW)
```

`(PRINT 'HELLO FOO)` prints in green against a yellow background.

Operating the Cursor on the Color Screen

To move the cursor to the color screen, slide the cursor off the left or right edge of the black and white screen onto the color screen, or call the `CURSORPOSITION` or `CURSORSCREEN` function.

`(WORPCURSOR` *ENABLE*`)` [Function]

If *ENABLE* is NIL, you cannot exit the current screen by sliding the cursor off.

`(CURSORPOSITION` *NEWPOSITION* `--)` [Function]

NEWPOSITION can be a position or a screenposition.

`(CURSORSCREEN` *SCREEN XCOORD YCOORD*`)` [Function]

Moves the cursor to the screenposition determined by *SCREEN*, *XCOORD*, and *YCOORD*. *SCREEN* should be the value of either `(COLORSCREEN)` or `(MAINSCREEN)`.

While on the color screen, the cursor is placed by doing `BITBLT`s in software rather than with microcode and hardware as with the black and white cursor. It is automatically taken down whenever an operation is performed that changes any bits on the color screen. The speed of the color cursor compares well with that of the black–and–white cursor but there can be a noticeable flicker when there is much input/output to the color screen. While the cursor is on the color screen, the black-and-white cursor is cleared giving the appearance that there is never more than one cursor at a given time.

Miscellaneous Color Functions

`(COLORIZEBITMAP` *BITMAP 0COLOR 1COLOR BITSPERPIXEL*`)` [Function]

Creates a color bitmap from a black–and–white bitmap. The returned bitmap has color number *1COLOR* in those pixels of *BITMAP* that were 1 and *0COLOR* in those pixels of *BITMAP* that were 0. This provides a way of producing a color bitmap from a black–and –white bitmap.

`(UNCOLORIZEBITMAP` *BITMAP COLORMAP*`)` [Function]

Creates a black–and–white bitmap from a color bitmap.

`(SHOWCOLORTESTPATTERN` *BARSIZE*`)` [Function]

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

[This page intentionally left blank]

COLOR

Introduction

This document describes software for driving color displays. In order to run COLOR, you need either a Sun (3 or 4) with CG4 color hardware and display, a Dorado (Xerox 1132) with attached color display, or a Dandelion (Xerox 1108) with attached BusMaster and color display.

The color software which is distributed among a number of files can be divided into a machine independent group of files that all users can usefully load and a machine dependent group containing files that work for particular combinations of hardware.

The machine independent color graphics code is stored in the library files LLCOLOR.LCOM and COLOR.LCOM. LOADing COLOR.LCOM causes LLCOLOR.LCOM to be LOADed.

The machine dependent portions of Xerox Lisp color software is stored in files such as MAIKOCOLOR.LCOM, DORADOCOLOR.LCOM, or COLORNNCC.LCOM. The user LOADs one of these files according to what kind of machine and color card the user is using.

The Sun color driver resides in the file MAIKOCOLOR.LCOM which loads LLCOLOR.LCOM and COLOR.LCOM. The CG4 device suppports 8 bpp at 1152 by 900 resolution. The user must be running ldecolor, the special color capable emulator. The physical display monitor is shared by both the monochrome and color screens (described below) .

The Dorado color driver resides in the file DORADOCOLOR.LCOM which loads LLCOLOR.LCOM and COLOR.LCOM. The Dorado color board supports four or eight bits per pixel (bpp) at 640 by 480 resolution. (The board supports 24 bpp also, but Xerox Lisp doesn't yet.)

The Dandelion color drivers reside in the files DANDELIONUFO.LCOM, DANDELIONUFO4096.LCOM, and COLORNNCC.LCOM, one package for each of three different kinds of boards. The user should load one of these packages on a Dandelion attached to a BusMaster and color display. The DANDELIONUFO and DANDELIONUFO4096 packages drive 4 bpp at 640 by 400 resolution color boards used inside Xerox which have been made obsolete by COLORNNCC. The COLORNNCC package drives an 8 bpp color at 512 by 480 resolution board, the Revolution 512 x 8, made by Number Nine Computer Corporation. The Revolution 512 x 8 is available both inside and outside Xerox through Number Nine.

Hardware Displays and Software Screens

On some workstations (such as the Dorado and Dandelion), there may be physically two separate displays. On most Suns, there is a single physical display, which additionally may be shared by two Unix devices. One device is monochrome (b/w), and the other is color.

To support the various hardware configurations and external display devices, the software has a special datatype, a "screen". There are two distinct instances of screens, a b/w screen, and a color screen. A screen represents and controls a physical hardware display, and contains windows, icons, and tracks the mouse.

On workstations with physically two separate hardware displays, each display is represented by a corresponding screen data structure. On workstations with a single hardware display, the display is shared by both the b/w screen and the color screen.

In all cases, before initialization only the b/w screen (and thus display) is visible and active. After initialization both screens are active (can contain screen images), although on single displays, only one screen is visible at a time. Since each screen logically controls a display, we will henceforth use the terms "screen" and "display" interchangeably. Screens are discussed in greater detail below.

Turning the Color Display Software On and Off

The color display software can be turned on and off. While the color display software is on, the memory used for the color display screen bitmap is locked down, and a small amount of processing time is used to drive the color display.

(COLORDISPLAYP) [Function]

returns T if the color display is on; otherwise it returns NIL.

(COLORDISPLAY*ONOFF TYPE***)** [Function]

turns off the color display if *ONOFF* is ’OFF. If *ONOFF* is ’ON, it turns on the color display allocating memory for the color screen bitmap. *TYPE* should be one of ’MAIKOCOLOR, ’DORADOCOLOR, ’DANDELIONUFO, ’DANDELIONUFO4096, or ’COLORNNCC. The usual sequence of events for the user is to LOAD the software needed to drive a particular color card and then to call COLORDISPLAY with the appropriate *TYPE* to turn the software on. For example,

```
(LOAD ’COLOR.LCOM)
(LOAD ’COLORNNCC.LCOM)
(COLORDISPLAY ’ON ’REV512X8)
```

will turn on the software needed to drive the Number Nine Computer Corporation’s Revolution 512 x 8 card with 1108 and BusMaster.

Besides initializing or reinitializing a color card that has been powered off, COLORDISPLAY allocates memory for the color screen bitmap. Turning on the color display requires allocating and locking down the memory necessary to hold the color display screen bitmap. Turning off the color display frees this memory.

Colors

The number of bits per pixel determines the number of different colors that can be displayed at one time. When there are 4 bpp, 16 colors can be displayed at once. When there are 8 bpp, 256 colors can be displayed at once. A table called a *color map* determines what color actually appears for each pixel value. A color map gives the color in terms of how much of the three primary colors (red, green, and blue) is displayed on the screen for each possible pixel value.

A color can be represented as a number, an atom, or a triple of numbers. Colors are ultimately given their final interpretation into how much red, blue, and green they represent through a color map. A color map maps a color number ([0 . . . 2^{nbits}-1]) into the intensities of the three color guns (primary colors red, green, and blue). Each entry in the color map has eight bits for each of the primary colors, allowing 256 levels per primary or 2^{24} possible colors (not all of which are distinct to the human eye). Within Xerox Lisp programs, colors can be manipulated as numbers, red-green-blue triples, names, or hue-lightness-saturation triples. Any function that takes a color accepts any of the different representations.

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

Red Green Blue Triples

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

RGB [Record]

is a record that is defined as (RED GREEN BLUE); it can be used to manipulate RGB triples.

COLORNAMES [Association list]

maps names into colors. The CDR of the color name’s entry is used as the color corresponding to the color name. This can be any of the other representations. (Note: It can even be another color name. Loops in the name space such as would be caused by putting ’(RED . CRIMSON) and ’(CRIMSON . RED) on COLORNAMES are *not* checked for by the system.) Some color names are available in the initial system and are intended to allow color programs written by different users to coexist. These are:

Name	RGB	Number in default color maps	
BLACK	(0 0 0)	15	255
BLUE	(0 0 255)	14	252
GREEN	(0 255 0)	13	227
CYAN	(0 255 255)	12	224
RED	(255 0 0)	3	31
MAGENTA	(255 0 255)	2	28
YELLOW	(255 255 0)	1	3
WHITE	(255 255 255)	0	0

Hue Lightness Saturation Triples

A hue lightness saturation triple is a list of three numbers. The first number (HUE) is an integer between 0 and 355 and indicates a position in degrees on a color wheel (blue at 0, red at 120, and green at 240). The second (LIGHTNESS) is a real number between zero and one that indicates how much total intensity is in the color. The third (SATURATION) is a real number between zero and one that indicates how disparate the three primary levels are.

HLS [Record]

is a record defined as (HUE LIGHTNESS SATURATION); it is provided to manipulate HLS triples. Example: the color blue is represented in HLS notation by (0 .5 1.0).

(COLORNUMBERP *COLOR BITSPERPIXEL NOERRFLG***)**[Function
]

returns the color number (offset into the screen color map) of *COLOR*. *COLOR* is one of the following:

· A positive number less than the maximum number of colors,

· A color name,

· AN RGB triple, or

· An HLS triple.

If *COLOR* is one of the above and is found in the screen color map, its color number in the screen color map is returned. If not, an error is generated unless NOERRFLG is non-NIL, in which case NIL is returned.

(RGBP *X***)** [Function]

returns *X* if *X* is an RGB triple; NIL otherwise.

(HLSP *X***)** [Function]

returns *X* if *X* is an HLS triple; NIL otherwise.

Color Maps

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

(SCREENCOLORMAP *NEWCOLORMAP***)** [Function]

reads and sets the color map that is used by the color display. If *NEWCOLORMAP* is non-NIL, it should be a color map, and **SCREENCOLORMAP** sets the system color map to be that color map. The value returned is the value of the screen color map before **SCREENCOLORMAP** was called. If *NEWCOLORMAP* is NIL, the current screen color map is returned without change.

(CMYCOLORMAP CYANBITS MAGENTABITS YELLOWBITS BITSPERPIXEL**)** [Function]

Returns a color map that assumes the BITSPERPIXEL bits are to be treated as three separate color planes with CYANBITS bits being in the cyan plane, MAGENTABITS bits being in the magenta plane, and YELLOWBITS bits being in the yellow plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 0 and black is 255.

(RGBCOLORMAP REDBITS GREENBITS BLUEBITS BITSPERPIXEL**)** [Function]

Returns a color map that assumes the BITSPERPIXEL bits are to be treated as three separate color planes with REDBITS bits being in the red plane, GREENBITS bits being in the green plane, and BLUEBITS bits being in the blue plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 255 and black is 0.

(GRAYCOLORMAP BITSPERPIXEL**)** [Function]

Returns a color map containing shades of gray. White is 0 and black is 255.

(COLORMAPCREATE *INTENSITIES BITSPERPIXEL***)** [Function]

creates a color map for a screen that has *BITSPERPIXEL* bits per pixel. If *BITSPERPIXEL* is NIL, the number of bits per pixel is taken from the current color display setting. *INTENSITIES* specifies the initial colors that should be in the map. If *INTENSITIES* is not NIL, it should be a list of color specifications other than color numbers, e.g., the list of RGB triples returned by the function INTENSITIESFROMCOLOR MAP.

(INTENSITIESFROMCOLORMAP *COLORMAP***)** [Function]

returns a list of the intensity levels of *COLORMAP* (default is (SCREENCOLORMAP)) in a form accepted by COLORMAPCREATE. This list can be written on file and thus provides a way of saving color map specifications.

(COLORMAPCOPY *COLORMAP BITSPERPIXEL***)** [Function]

returns a color map that contains the same color intensities as *COLORMAP* if *COLORMAP* is a color map. Otherwise, it returns a color map with default color values.

(MAPOFACOLOR *PRIMARIES***)** [Function]

returns a color map that is different shades of one or more of the primary colors. For example, (MAPOFACOLOR ’(RED GREEN BLUE)) gives a color map of different shades of gray; (MAPOFACOLOR ’RED) gives different shades of red.

Changing Color Maps

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

(SETCOLORINTENSITY *COLORMAP COLORNUMBER COLORSPEC***)** [Function]

sets the primary intensities of color number *COLORNUMBER* in the color map *COLORMAP* to the ones specified by *COLORSPEC*. *COLORSPEC* can be either an RGB triple, an HLS triple, or a color name. The value returned is NIL.

(COLORLEVEL *COLORMAP COLORNUMBER PRIMARY NEWLEVEL***)** [Function]

sets and reads the intensity level of the primary color *PRIMARY* (RED, GREEN, or BLUE) for the color number *COLORNUMBER* in the color map *COLORMAP*. If *NEWLEVEL* is a number between 0 and 255, it is set. The previous value of the intensity of *PRIMARY* is returned.

(ADJUSTCOLORMAP *PRIMARY DELTA COLORMAP***)** [Function]

adds *DELTA* to the intensity of the *PRIMARY* color value (RED, GREEN, or BLUE) for every color number in *COLORMAP*.

(ROTATECOLORMAP *STARTCOLOR THRUCOLOR***)** [Function]

rotates a sequence of colors in the SCREENCOLORMAP. The rotation moves the intensity values of color number *STARTCOLOR* into color number *STARTCOLOR*+1, the intensity values of color number *STARTCOLOR*+1 into color number *STARTCOLOR*+2, etc., and *THRUCOLOR's* values into *STARTCOLOR*.

(EDITCOLORMAP *VAR NOQFLG***)** [Function]

allows interactive editing of a color map. If *VAR* is an atom whose value is a color map, its value is edited. Otherwise a new color map is created and edited. The color map being edited is made the screen color map while the editing takes place so that its effects can be observed. The edited color map is returned as the value. If *NOQFLG* is NIL and the color display is on, you are asked if you want a test pattern of colors. A yes response causes the function SHOWCOLORTESTPATTERN to be called, which displays a test pattern with blocks of each of the possible colors.

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

The color being edited can also be changed by selecting the menu item "PickPt." This switches the cursor onto the color screen and allows you to select a point from the color screen. It then edits the color of the selected point.

To stop the editing, move the cursor into the title of the editing window and press the middle button. This brings up a menu. Select Stop to quit.

Color Bitmaps

A color bitmap is actually a bitmap that has more than one bit per pixel. To test whether a bitmap is a color bitmap, the function BITSPERPIXEL can be used.

(BITSPERPIXEL *BITMAP***)** [Function]

returns the bits per pixel of *BITMAP*; if this does not equal one, *BITMAP* is a color bitmap.

In multiple-bit-per-pixel bitmaps, the bits that represent a pixel are stored contiguously. BITMAPCREATE is passed a *BITSPERPIXEL* argument to create multiple-bit-per-pixel bitmaps.

(BITMAPCREATE *WIDTH HEIGHT BITSPERPIXEL***)** [Function]

creates a color bitmap that is *WIDTH* pixels wide by *HEIGHT* pixels high allowing *BITSPERPIXEL* bits per pixel. Currently any value of *BITSPERPIXEL* except one, four, eight, or NIL (defaults to one) causes an error.

A four-bit-per-pixel color screen bitmap uses approximately 76K words of storage, and an eight-bit-per-pixel one uses approximately 153K words. There is only one such bitmap. The following function provides access to it.

(COLORSCREENBITMAP) [Function]

returns the bitmap that is being or will be displayed on the color display. This is NIL if the color display has never been turned on (see COLORDISPLAY below).

Screens, Screenpositions, and Screenregions

In addition to positions and regions, the user needs to be aware of screens, screenpositions, and screenregions in the presence of multiple screens.

Screens

SCREEN [Datatype]

There are generally two screen datatype instances in existence when working with color. This is because the user is attached to two displays, a black and white display and a color display.

(MAINSCREEN) [Function]

returns the screen datatype instance that represents the black and white screen. This will be something like {SCREEN}#74,24740.

(COLORSCREEN) [Function]

returns the screen datatype instance that represents the color screen. Screens appear as part of screenpositions and screenregions, serving as the extra information needed to make clear whether a particular position or region should be viewed as lying on the black and white display or the color display.

(SCREENBITMAP *SCREEN***)** [Function]

returns the bitmap destination of *SCREEN*. If *SCREEN*=NIL, returns the black and white screen bitmap.

Screenpositions

SCREENPOSITION [Record]

Somewhat like a position, a screenposition denotes a point in an X,Y coordinate system on a particular screen. Screenpositions have been defined according to the following record declaration:

(RECORD SCREENPOSITION (SCREEN . POSITION)

(SUBRECORD POSITION))

A SCREENPOSITION is an instance of a record with fields XCOORD, YCOORD, and SCREEN and is manipulated with the standard record package facilities. For example, (create SCREENPOSITION XCOORD _ 10 YCOORD _ 20 SCREEN _ (COLORSCREEN)) creates a screenposition representing the point (10,20) on the color display. The user can extract the position of a screenposition by fetching its POSITION. For example, (fetch (SCREENPOSITION POSITION) of SP12).

Screenregions

SCREENREGION [Record]

Somewhat like a region, a screenregion denotes a rectangular area in a coordinate system. Screenregions have been defined according to the following record declaration:

(RECORD SCREENREGION (SCREEN . REGION)

(SUBRECORD REGION))

Screenregions are characterized by the coordinates of their bottom left corner and their width and height. A SCREENREGION is a record with fields LEFT, BOTTOM, WIDTH, HEIGHT, and SCREEN. It can be manipulated with the standard record package facilities. There are access functions for the REGION record that return the TOP and RIGHT of the region. The user can extract the region of a screenregion by fetching its REGION. For example, (fetch (SCREENREGION REGION) of SR8).

Screenposition and Screenregion Prompting

The following functions can be used by programs to allow the user to interactively specify screenpositions or screenregions on a display screen.

(GETSCREENPOSITION *WINDOW CURSOR***)** [Function]

Similar to GETPOSITION. Returns a SCREENPOSITION that is specified by the user. GETSCREENPOSITION waits for the user to press and release the left button of the mouse and returns the cursor screenposition at the time of release. If *WINDOW* is a WINDOW, the screenposition will be on the same screen as *WINDOW* and in the coordinate system of *WINDOW*'s display stream. If *WINDOW* is NIL, the position will be in screen coordinates.

(GETBOXSCREENPOSITION *BOXWIDTH BOXHEIGHT ORGX ORGY WINDOW PROMPTMSG***)** [Function]

Similar to GETBOXPOSITION. Returns a SCREENPOSITION that is specified by the user. Allows the user to position a "ghost" region of size *BOXWIDTH* by *BOXHEIGHT* on a screen, and returns the SCREENPOSITION of the lower left corner of the screenregion chosen. A ghost region is locked to the cursor so that if the cursor is moved, the ghost region moves with it. The user can change to another corner by holding down the right button. With the right button down, the cursor can be moved across a screen or to other screens without effect on the ghost region frame. When the right button is released, the mouse will snap to the nearest corner, which will then become locked to the cursor. (The held corner can be changed after the left or middle button is down by holding both the original button and the right button down while the cursor is moved to the desired new corner, then letting up just the right button.) When the left or middle button is pressed and released, the lower left corner of the screenregion chosen at the time of release is returned. If *WINDOW* is a WINDOW, the screenposition will be on the same screen as *WINDOW* and in the coordinate system of *WINDOW*'s display stream. If *WINDOW* is NIL, the position will be in screen coordinates.its lower left corner in screen coordinates.

(GETSCREENREGION *MINWIDTH MINHEIGHT OLDREGION NEWREGIONFN NEWREGIONFNARG INITCORNERS***)** [Function]

Similar to GETREGION. Returns a SCREENREGION that is specified by the user. Lets the user specify a new screenregion and returns that screenregion. GETSCREENREGION prompts for a screenregion by displaying a four-pronged box next to the cursor arrow at one corner of a "ghost" region: . If the user presses the left button, the corner of a "ghost" screenregion opposite the cursor is locked where it is. Once one corner has been fixed, the ghost screenregion expands as the cursor moves.

To specify a screenregion: (1) Move the ghost box so that the corner opposite the cursor is at one corner of the intended screenregion. (2) Press the left button. (3) Move the cursor to the screenposition of the opposite corner of the intended screenregion while holding down the left button. (4) Release the left button.

Before one corner has been fixed, one can switch the cursor to another corner of the ghost screenregion by holding down the right button. With the right button down, the cursor changes to a "forceps" () and the cursor can be moved across a screen or to other screens without effect on the ghost screenregion frame. When the right button is released, the cursor will snap to the nearest corner of the ghost screenregion.

After one corner has been fixed, one can still switch to another corner. To change to another corner, continue to hold down the left button and hold down the right button also. With both buttons down, the cursor can be moved across a screen or to other screens without effect on the ghost screenregion frame. When the right button is released, the cursor will snap to the nearest corner, which will become the moving corner. In this way, the screenregion may be moved all over a screen and to other screens, before its size and screenposition is finalized.

The size of the initial ghost screenregion is controlled by the *MINWIDTH*, *MINHEIGHT*, *OLDREGION*, and *INITCORNERS* arguments.

(GETBOXSCREENREGION *WIDTH HEIGHT ORGX ORGY WINDOW PROMPTMSG***)** [Function]

Similar to GETBOXREGION. Returns a SCREENREGION that is specified by the user. Performs the same prompting as GETBOXSCREENPOSITION and returns the SCREENREGION specified by the user instead of the SCREENPOSITION of its lower left corner.

Color Windows and Menus

The Xerox Lisp window system provides both interactive and programmatic constructs for creating, moving, reshaping, overlapping, and destroying windows in such a way that a program can use a window in a relatively transparent fashion (see page X.XX). Menus are a special type of window provided by the window system, used for displaying a set of items to the user, and having the user select one using the mouse and cursor. The menu facility also allows users to create and use menus in interactive programs (see page X.XX). As of the LUTE release of Xerox Lisp, it is possible for the user to create and use windows and menus on the color display.

(CREATEW *REGION TITLE BORDERSIZE NOOPENFLG***)** [Function]

Creates a new window. *REGION* indicates where and how large the window should be by specifying the exterior screenregion of the window. In a user environment with multiple screens, such as a black and white screen and color screen both connected to the same machine, there is a new special problem in indicating which screen the *REGION* is supposed to be a region of. This problem is solved by allowing CREATEW to take screenregion arguments as *REGION*. For example,

```
(SETQ FOO (CREATEW (CREATE SCREENREGION
                                        SCREEN _
(COLORSCREEN)
                           LEFT _ 20
                           BOTTOM _ 210
                           WIDTH _ 290
                           HEIGHT _ 170)
                   "FOO WINDOW"))
```

creates a window titled "FOO WINDOW" on the color screen. To create a window on the black and white screen, the user should use SCREEN _ (MAINSCREEN) in the CREATE SCREENREGION expression. Note that it is still perfectly legal to pass in a *REGION* that is a region, not a screenregion, to CREATEW, but it is preferable to be passing screenregions rather than regions to CREATEW. This is because when *REGION* is a region, *REGION* is disambiguated in a somewhat arbitrary manner that may not always turn out to be what the user was hoping for.

When *REGION* is a region, *REGION* is disambiguated by coercing *REGION* to be a screenregion on the screen which currently contains the cursor. This is so that software calling CREATEW with regions instead of screenregions tends to do the right thing in a user environment with multiple screens.

(WINDOWPROP *WINDOW PROP NEWVALUE***)** [NoSpread Function]

If *PROP*='SCREEN, then WINDOWPROP returns the screen *WINDOW* is on. If *NEWVALUE* is given, (even if given as NIL), with *PROP*='SCREEN, then WINDOWPROP will generate an error. Any other *PROP* name is handled in the usual way.

(OPENWINDOWS *SCREEN***)** [Function]

Returns a list of all open windows on *SCREEN* if *SCREEN* is a screen datatype such as (MAINSCREEN) or (COLORSCREEN). If *SCREEN*=NIL, then *SCREEN* will default to the screen containing the cursor. If *SCREEN*=T, then a list of all open windows on all screens is returned.

Color Fonts

The user can create color fonts and specify in the font profile that certain color fonts be used when printing in color.

Color Font Creation

The user can create and manipulate color fonts through the same functions that are used to create and manipulate black and white fonts. This is made possible in some cases by there being new ways to call familiar font functions.

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

In addition to creating black and white fonts, FONTCREATE can be used to create color fonts. For example,

```
(FONTCREATE 'GACHA 10
                                  '(BOLD REGULAR REGULAR YELLOW
BLUE)
                  0 '8DISPLAY)
```

will create an 8 bit per pixel font with blue letters on a yellow background. The user indicates the color and bits per pixel of the font by the FACE and DEVICE arguments passed to FONTCREATE. DEVICE='8DISPLAY means to create an 8bpp font and DEVICE='4DISPLAY means to create a 4bpp font. A color font face is a 5 tuple,

```
(WEIGHT SLOPE EXPANSION BACKCOLOR FORECOLOR)
```

whereas a black and white font face is just a 3 tuple,

```
(WEIGHT SLOPE EXPANSION)
```

The FORECOLOR is the color of the characters of the font and the BACKCOLOR is the color of the background behind the characters that gets printed along with the characters. Both BACKCOLOR and FORECOLOR are allowed to a color name, color number, or any other legal color representation. A color font face can also be represented as a LITATOM. A three character atom such as MRR or any of the special atoms STANDARD, ITALIC, BOLD, BOLDITALIC can optionally be continued by hyphenating on BACKCOLOR and FORECOLOR suffixes. For example, MRR-YELLOW-BLUE

BOLD-YELLOW-RED

ITALIC-90-200

BRR-100-53

are acceptable color font faces. Hence,

(FONTCREATE 'GACHA 10 'BOLD-YELLOW-BLUE 0 '8DISPLAY)

will create a color font. LITATOM FACE arguments fall into one of the following patterns:

wse	wse-backcolor-forecolor
STANDARD	STANDARD-backcolor-forecolor
ITALIC	ITALIC-backcolor-forecolor
BOLD	BOLD-backcolor-forecolor
BOLDITALIC	BOLDITALIC-backcolor-forecolor

where w=B, M, or L; s=I or R; e=R, C, or E; backcolor=a color name or color number; and forecolor=a color name or color number.

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

Returns the value of the *PROP* property of font *FONT*. Besides black and white font properties, the following font properties are recognized:

FORECOLOR The color of the characters of the font, represented as a color number. This is the color in which the characters of the font will print.

BACKCOLOR The color of the background of the characters of the font, represented as a color number. This is the color in which the the background of characters of the font will print. A font with red characters on a yellow background would have a red FORECOLOR and a yellow BACKCOLOR.

Color Font Profiles

Font profiles are the facility PRETTYPRINT uses to print different elements (user functions, system functions, clisp words, comments, etc.) in different fonts to emphasize (or deemphasize) their importance, and in general to provide for a more pleasing appearance. The user can specify that different colors of fonts be used for different kinds of elements when printing in color. A well chosen font profile will allows user to DEDIT functions, PP functions, and SEE source files in color, for example. A **FONTPROFILE** such as

```
((DEFAULTFONT 1 (GACHA 10)
              (GACHA 8)
              (TERMINAL 8)
              (4DISPLAY (GACHA 10 MRR-WHITE-RED))
              (8DISPLAY (GACHA 10 MRR-WHITE-RED)))
 (BOLDFONT 2 (HELVETICA 10 BRR)
            (HELVETICA 8 BRR)
            (MODERN 8 BRR)
            (4DISPLAY (HELVETICA 10 BRR-WHITE-MAGENTA))
            (8DISPLAY (HELVETICA 10 BRR-WHITE-MAGENTA)))
 (LITTLEFONT 3 (HELVETICA 8)
            (HELVETICA 6 MIR)
            (MODERN 8 MIR)
            (4DISPLAY (HELVETICA 8 MRR-WHITE-GREEN))
            (8DISPLAY (HELVETICA 8 MRR-WHITE-GREEN)))
 (BIGFONT 4 (HELVETICA 12 BRR)
       (HELVETICA 10 BRR)
       (MODERN 10 BRR)
       (4DISPLAY (HELVETICA 12 BRR-WHITE-BLUE))
       (8DISPLAY (HELVETICA 12 BRR-WHITE-BLUE)))
 (USERFONT BOLDFONT)
 (COMMENTFONT LITTLEFONT)
 (LAMBDAFONT BIGFONT)
 (SYSTEMFONT)
 (CLISPFONT BOLDFONT)
 ...)
```

would have comments print in green and clisp words print in blue while ordinairy atoms would print in red.

Not all combinations of fonts will be aesthetically pleasing and the user may have to experiment to find a compatible set.

The user should indicate what font is to be used for each font class by calling the function FONTPROFILE:

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

Sets up the font classes as determined by *PROFILE*, a list of elements which defines the correspondence between font classes and specific fonts. Each element of *PROFILE* is a list of the form:

(*FONTCLASS FONT# DISPLAYFONT PRESSFONT INTERPRESSFONT* **(***OTHERDEVICE1 OTHERFONT1***)** ... **(***OTHERDEVICEn OTHERFONTn***))**

FONTCLASS is the font class name and *FONT#* is the font number for that class. *DISPLAYFONT*, *PRESSFONT*, and *INTERPRESSFONT* are font specifications (of the form accepted by FONTCREATE) for the fonts to use when printing to the black and white display and to Press and Interpress printers respectively. The appearance of color fonts can be affected by including an **(***OTHERDEVICEi OTHERFONTi***)** entry where *OTHERDEVICEi* is either 4DISPLAY or 8DISPLAY for a 4 bits per pixel or 8 bits per pixel color font and *OTHERFONTi* is a color font specification such as (GACHA 10 MRR-WHITE-RED).

FONTPROFILE [Variable]

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

Using Color

The current color implementation allows display streams to operate on color bitmaps. The two functions DSPCOLOR and DSPBACKCOLOR set the color in which a stream draws when the user defaults a color argument to a drawing function.

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

sets the foreground color of a stream. It returns the previous foreground color. If *COLOR* is NIL, it returns the current foreground color without changing anything. The default foreground color is MINIMUMCOLOR=0, which is white in the default color maps.

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

sets the background color of a stream. It returns the previous background color. If *COLOR* is NIL, it returns the current background color without changing anything. The default background color is (MAXIMUMCOLOR BITSPERPIXEL)=15 or 255, which is black in the default color maps.

The BITBLT, line-drawing routines, and curve-drawing routines routines know how to operate on a color-capable stream. Following are some notes about them.

BITBLTing in Color

If BITBLTing from a color bitmap onto another color bitmap of the same bpp, the operations PAINT, INVERT, and ERASE are done on a bit level, not on a pixel level. Thus painting color 3 onto color 10 results in color 11.

If BITBLTing from a black-and-white bitmap onto a color bitmap, the one bits appear in the DSPCOLOR, and the zero bits in DSPBACKCOLOR. BLTing from black-and-white to color is fairly expensive; if the same bitmap is going to be put up several times in the same color, it is faster to create a color copy and then BLT the color copy.

If the source type is TEXTURE and the destination bitmap is a color bitmap, the *Texture* argument is taken to be a color. Thus to fill an area with the color BLUE assuming COLORSTR is a stream whose destination is the color screen, use (BITBLT NIL NIL NIL COLORSTR 50 75 100 200 ’TEXTURE ’REPLACE ’BLUE).

Drawing Curves and Lines in Color

For the functions DRAWCIRCLE, DRAWELLIPSE, and DRAWCURVE, the notion of a brush has been extended to include a color. A BRUSH is now (BRUSHSHAPE BRUSHSIZE BRUSHCOLOR). Also, a brush can be a bitmap (which can be a color bitmap).

Line-drawing routines take a color argument which is the color the line is to appear in if the destination of the display stream is a color bitmap.

(DRAWLINE *X1 Y1 X2 Y2 WIDTH OPERATION STREAM COLOR***)** [Function]

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

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

(DRAWBETWEEN *POS1 POS2 WIDTH OPERATION STREAM COLOR***)** [Function]

If the *COLOR* argument is NIL, the DSPCOLOR of the stream is used.

Printing in Color

Printing only works in REPLACE mode. The characters have a background color and a foreground color determined by the font face of the font the characters are being printed with.

Example of printing to an 8bpp color screen:

```
(SETQ FOO (CREATEW (CREATE SCREENREGION
                                        SCREEN _
(COLORSCREEN)
                              LEFT _ 20
                              BOTTOM _ 210
                              WIDTH _ 290
                              HEIGHT _ 170)
                    "FOO WINDOW"))
(DSPFONT (FONTCREATE 'GACHA
                     10
                     'MRR-YELLOW-GREEN
                     0
                     '8DISPLAY)
     FOO)
(PRINT 'HELLO FOO)        ; will print in green against a yellow
background.
```

Operating the Cursor on the Color Screen

The cursor can be moved to the color screen. The cursor can be moved to the color screen by sliding the cursor off the left or right edge of the black and white screen on to the color screen or by calling function CURSORPOSITION or CURSORSCREEN.

(CURSORPOSITION *NEWPOSITION* **- -)** [Function]

NEWPOSITION can be a position or a screenposition.

(CURSORSCREEN *SCREEN XCOORD YCOORD***)** [Function]

Moves the cursor to the screenposition determined by *SCREEN*, *XCOORD*, and *YCOORD*. *SCREEN* should be the value of either (COLORSCREEN) or (MAINSCREEN).

While on the color screen, the cursor is placed by doing BITBLTs in software rather than with microcode and hardware as with the black and white cursor. It is automatically taken down whenever an operation is performed that changes any bits on the color screen. The speed of the color cursor compares well with that of the black and white cursor but there can be a noticeable flicker when there is much input/output to the color screen. While the cursor is on the color screen, the black-and-white cursor is cleared giving the appearance that there is never more than one cursor at a given time.

Miscellaneous Color Functions

(COLORIZEBITMAP *BITMAP 0COLOR 1COLOR BITSPERPIXEL***)**[Function]

creates a color bitmap from a black and white bitmap. The returned bitmap has color number *1COLOR* in those pixels of *BITMAP* that were one and *0COLOR* in those pixels of *BITMAP* that were zero. This provides a way of producing a color bitmap from a black and white bitmap.

(UNCOLORIZEBITMAP *BITMAP COLORMAP***)** [Function]

creates a black and white bitmap from a color bitmap.

(SHOWCOLORTESTPATTERN *BARSIZE***)** [Function]

displays a pattern of colors on the color display. This is useful when editing a color map. The pattern has squares of the 16 possible colors laid out in two rows at the top of the screen. Colors 0 through 7 are in the top row, and colors 8 through 15 are in the next row. The bottom part of the screen is filled with bars of *BARSIZE* width with consecutive color numbers. The pattern is designed so that every color has a border with every other color (unless *BARSIZE* is too large to allow room for every color—about 20).

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.)

Note: You must use *s if you want to name files. If you do not, the `COPYFILES` code assumes that any *DESTINATION* specified without *s in it is a destination directory.

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)
```

Copies 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)
```

Uses `RENAMEFILE` instead.

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

Copies the files `TEST` and `WEST` from `{DSK}` to `{PHYLUM}<MYDIR>`.

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

Copies 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))
```

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

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

Copies all files on `{DSK}` that have no file name extensions to `{FLOPPY}`.

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

Makes `{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.

[This page intentionally left blank]

DATABASEFNS

DataBaseFns makes the construction and maintenance of MasterScope database 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 database, and loading the file also loads the database for you to use.

For example,

```
(LOAD 'FILE1)
```

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

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

The database is 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 analyzes any new or changed functions on the file, and a database for all of the functions on the file is written on a separate file whose name is of the form FILE.DATABASE. Whenever a file that has a database property with value YES is loaded via LOAD or LOADFROM, then the corresponding database file, if any, is also loaded. The database is not 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 database:

```
(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 database:

```
(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 database file is loaded:

`Yes`	Always try to load the database when a file is loaded.
`No`	Never try to load the database when a file is loaded.
`Ask`	(default) Ask whether to load the database when a file is loaded.

`SAVEDBFLG` [Variable]

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

`Yes`	Always try to save the database when a file is saved.
`No`	Never try to save the database when a file is saved.
`Ask`	(default) Ask whether to save the database when a file is saved.

`(DUMPDB` *FILE PROPFLG*`)` [Function]

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

`(LOADDB` *FILE ASKFLG*`)` [Function]

Loads the file `FILE.DATABASE` if one exists. After the database is loaded, the `DATABASE` property for `FILE` is set to `YES`, so that maintenance is automatic thereafter.

Database files include the date and full file name of the file to which they correspond. `LOADDB` prints out a warning message if it loads a database that does not correspond to the in-core version of the file, and asks you if you approve.

Note: `LOADDB` is the only approved way of loading a database. Attempting to `LOAD` a database 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

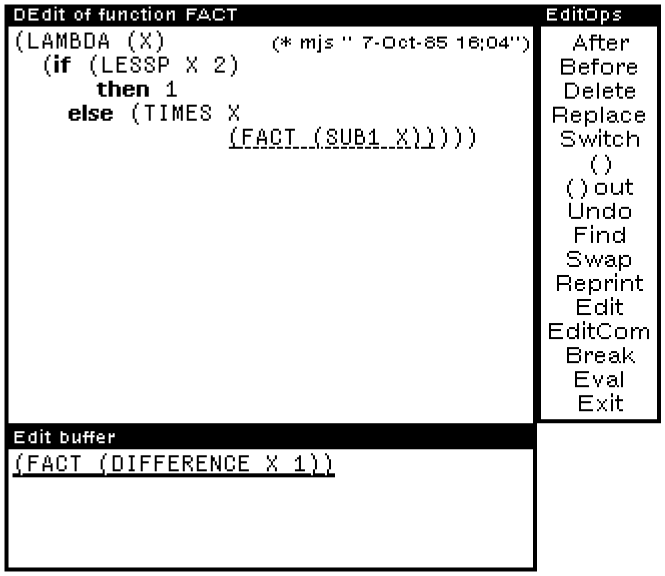
DEdit is normally called using one of the DEdit functions. See also "Advanced Features" below.

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:

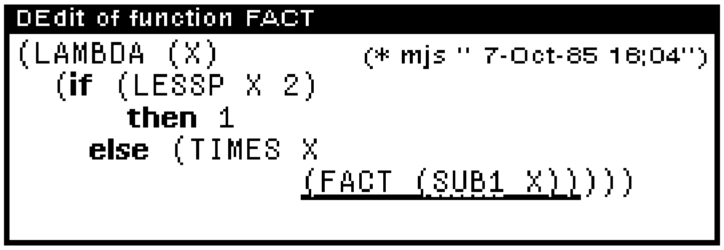
- The left button selects the object being directly pointed at.
- The middle button selects the containing list.

- 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) attaches 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 creates a type-in buffer window which positions itself under the current DEdit window and does 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 becomes the current selection (top of stack) and is 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 interprets the rest of the line as a teletype editor command that is interpreted when the line is closed. Likewise, Control-S `OLD NEW` substitutes `NEW` for `OLD`, and Control-F X finds 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

Invoke a DEdit command 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 brings 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 moves 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 the DEdit Parameters section 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 appears, 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 the Commands That Edit Parentheses section below).

`) IN` [DEdit Command]

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

`() OUT` [DEdit Command]

Removes parentheses from `TOP`.

`( OUT` [DEdit Command]

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

`) OUT` [DEdit Command]

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

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 the Calling DEdit section 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 the following:

```
(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.

[This page intentionally left blank]

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").

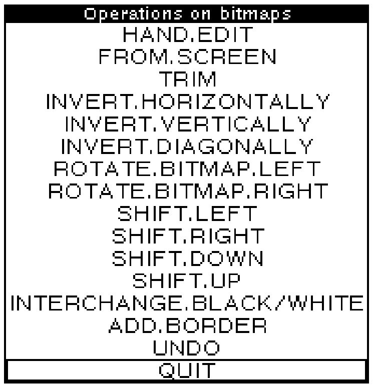
EDIT.BITMAP

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.

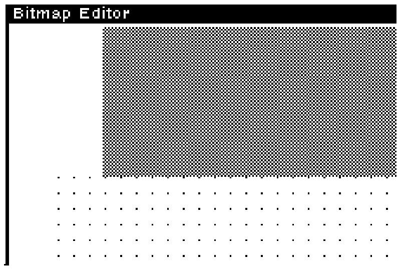
EditBitMap brings up a main menu containing the following items:



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.

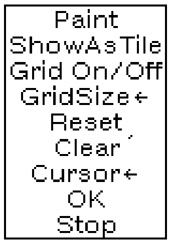
Window

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:

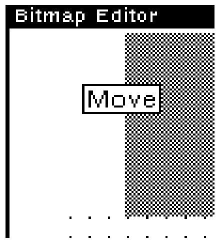


These menu items are described in the system prompt window when an item is selected.

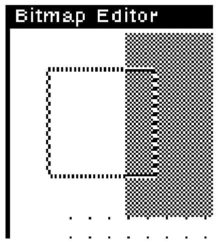
Mouse Buttons

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 corne. 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:

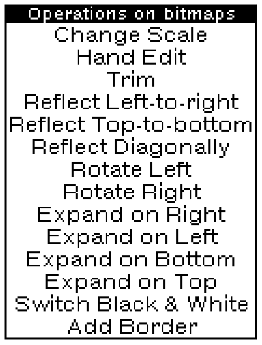
```
(SETQ NEWBITMAPNAME (EDITBM))
```

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]

Swaps black and white bits in a bitmap. On the EditBitMap menu, this is called by selecting `INTERCHANGE.BLACK/WHITE`. On the TEdit bitmap editor menu, it is called by selecting "Switch Black & 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.

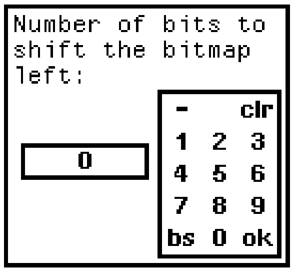
`(INTERACT&SHIFT.BITMAP.UP` *BITMAP*`)` [Function]

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.

[This page intentionally left blank]

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 Specifying What Files to Browse section 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 cannot 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 FileBrowser Commands section 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 FileBrowser Commands section below) and then use the RECOMPUTE command to be prompted for a pattern again.

Example

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

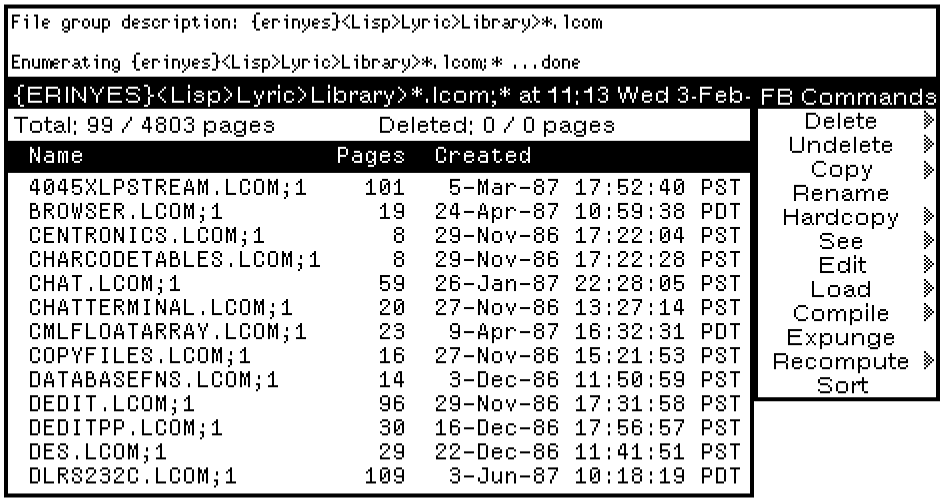
This topmost subwindow is where FileBrowser prints messages about what it is doing and receives input from you. Its contents are cleared before every command.

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
```

Figure 1. Tally Window

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 Selecting Files section 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 Getting Information About Files section below).

SCROLL bar

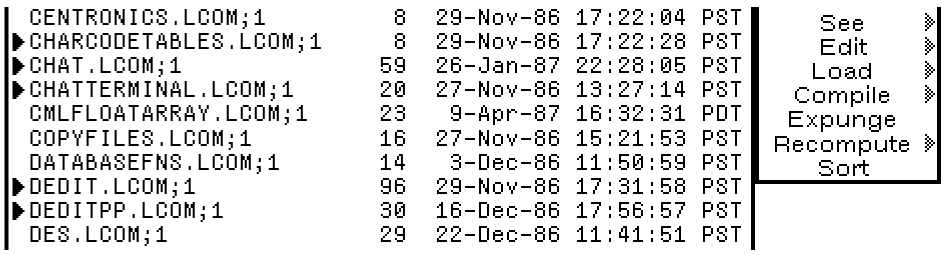
If there are more files in the listing than fit at one time in the browser window, you can scroll the browser window to view more files. Slide the mouse cursor out the left side of the browser window to get the scroll bar and press the left mouse button to scroll the region up and the right mouse button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

Figure 2. FileBrowser Window with Files Selected

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 Copy-Selecting Files section 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 Copy-Selecting Files section. 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".

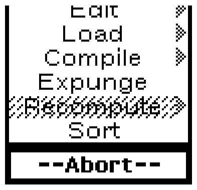
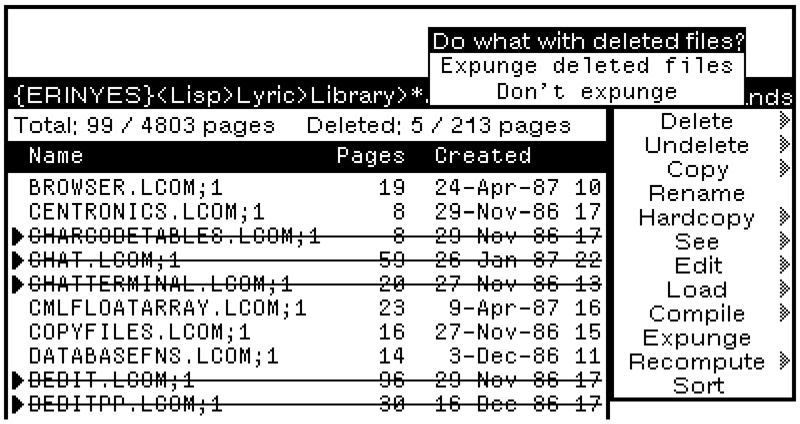


Figure 3. Command Menu with "Abort" Added at Bottom

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.



```
                                      Do what with deleted files?
                                       Expunge deleted files
{ERINYES}<Lisp>Lyric>Library>*.         Don't expunge          nds
Total: 99 / 4803 pages   Deleted: 5 / 213 pages    Delete    >
  Name                     Pages  Created          Undelete  >
  BROWSER.LCOM;1             19  24-Apr-87 10      Copy      >
  CENTRONICS.LCOM;1           8  29-Nov-86 17      Rename
▶ ~~CHARCODETABLES.LCOM;1       8  29-Nov-86 17~~      Hardcopy  >
▶ ~~CHAT.LCOM;1                59  26-Jan-87 22~~      See       >
▶ ~~CHATTERMINAL.LCOM;1        20  27-Nov-86 13~~      Edit      >
  CMLFLOATARRAY.LCOM;1       23   9-Apr-87 16      Load      >
  COPYFILES.LCOM;1           16  27-Nov-86 15      Compile   >
  DATABASEFNS.LCOM;1         14   3-Dec-86 11      Expunge
▶ ~~DEDIT.LCOM;1               96  29-Nov-86 17~~      Recompute >
▶ ~~DEDITPP.LCOM;1             30  16-Dec-86 17~~      Sort
```

Figure 4. FileBrowser Window with Files Selected for Deletion

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.

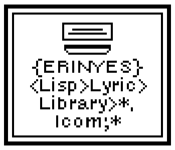


Figure 5. Shrink Icon with File Pattern

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.

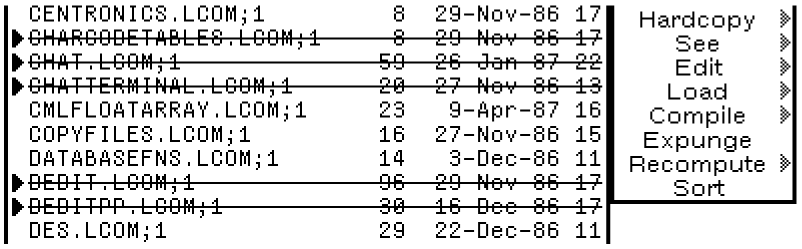
If the browser displays a large number of attributes, or your default printer font is too large, the listing may not accommodate all the attributes on one line, making the listing less readable. You may want to make the listing with fewer attributes, or use a smaller font for the listing (see description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

FileBrowser Commands

DELETE, UNDELETE

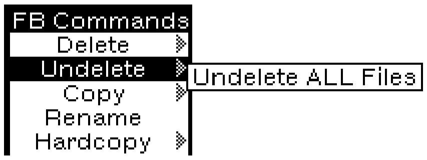
Removing a file from the file system using FileBrowser is a two-step process. First, mark the file or files for deletion. Then issue the Expunge command. Any time between the deletion and the expunge you can change your mind and undelete any of the files.

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.

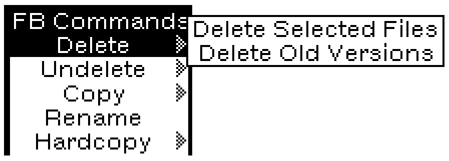


```
 CENTRONICS.LCOM;1         8  29-Nov-86 17
~~CHARCODETABLES.LCOM;1     8  29-Nov-86 17~~
~~CHAT.LCOM;1              59  26-Jan-87 22~~
~~CHATTERMINAL.LCOM;1      20  27-Nov-86 13~~
 CMLFLOATARRAY.LCOM;1     23   9-Apr-87 16
 COPYFILES.LCOM;1         16  27-Nov-86 15
 DATABASEFNS.LCOM;1       14   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).

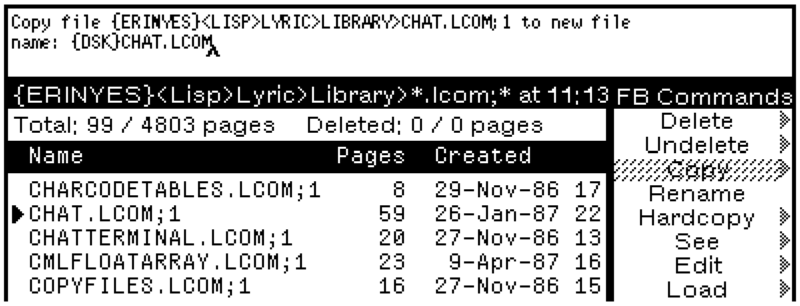
For example, if you wanted to copy only the most recent version of all the files in the browser to another location, you could do the following:

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
```

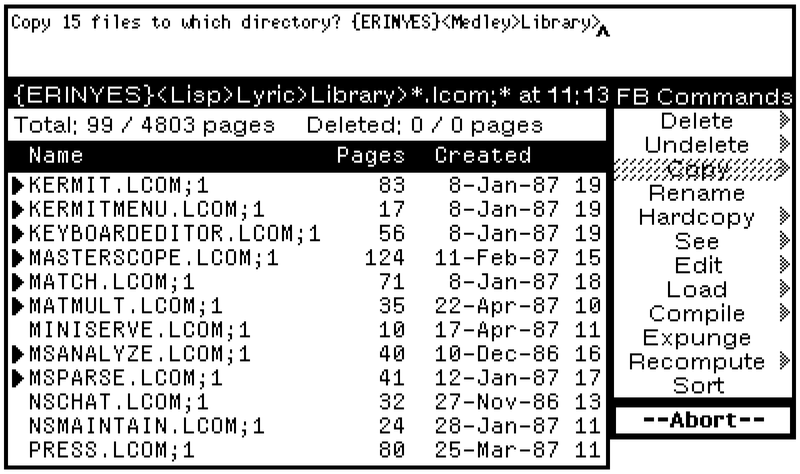
Note: Unless you specify a version number in the destination file name, the version number of the new file will be 1, or one higher than the highest existing version of the file in the destination directory, independent of the version number of the old name.

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).

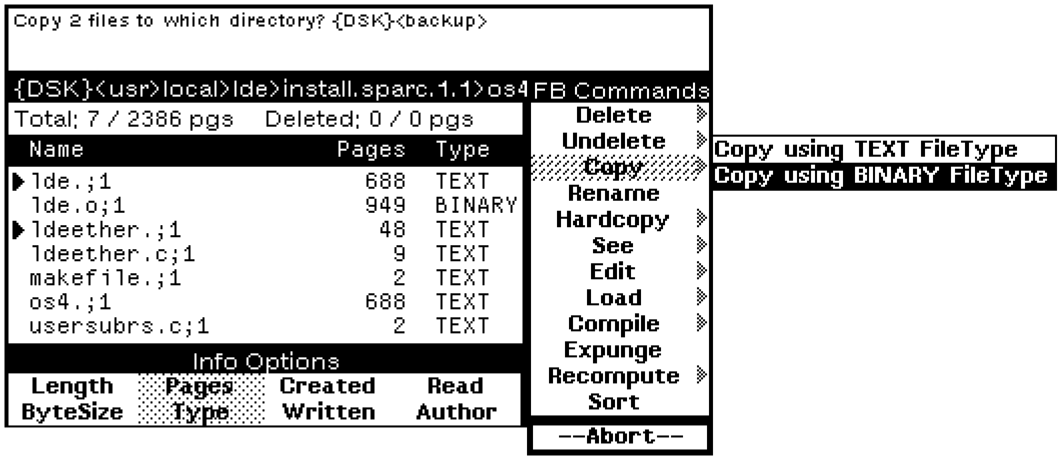


```
Copy 15 files to which directory? {ERINYES}<Medley>Library>

{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.

Copying files from filesystems that do not normally distinguish between **TEXT** and **BINARY** file types , like UNIX, can sometimes cause problems. On these systems the type of a file is inferred from its extension, or by using the Lisp variables **DEFAULTFILETYPE** and **DEFAULTFILETYPELIST**. At times, there will be files where the system cannot properly infer the type. By selecting the submenu of COPY you can copy files with the right attributes.



```
Copy 2 files to which directory? {DSK}<backup>

{DSK}<usr>local>lde>install.sparc.1.1>os4
Total: 7 / 2386 pgs    Deleted: 0 / 0 pgs
 Name                        Pages  Type
▶lde.;1                       688   TEXT
 lde.o;1                      949   BINARY
▶ldeether.;1                   48   TEXT
 ldeether.c;1                   9   TEXT
 makefile.;1                    2   TEXT
 os4.;1                       688   TEXT
 usersubrs.c;1                  2   TEXT
```

In this example, since the chosen files had no extension, the system incorrectly inferred their type from the value of **DEFAULTFILETYPE**. By selecting the **BINARY** copy option, the correct type of the file will be preserved after the operation.

RENAME

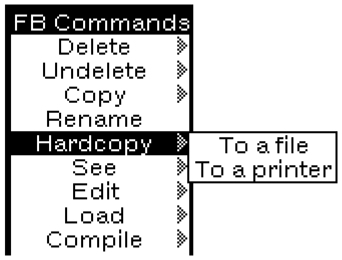
The RENAME command is used for changing the name of a file or group of files, or for moving a file or group of files to a different directory.

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.

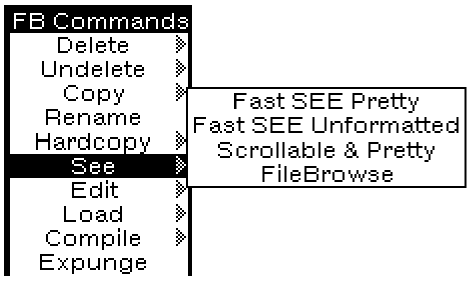
Note: For files stored on servers not supporting random access, FileBrowser is currently unable to determine that a file is in TEdit format unless the file has the extension `.TEDIT`. Therefore you should use TEdit to hardcopy TEdit files with other extensions. Use FileBrowser's EDIT command (to call 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 Getting Hardcopy Directory Listings section.

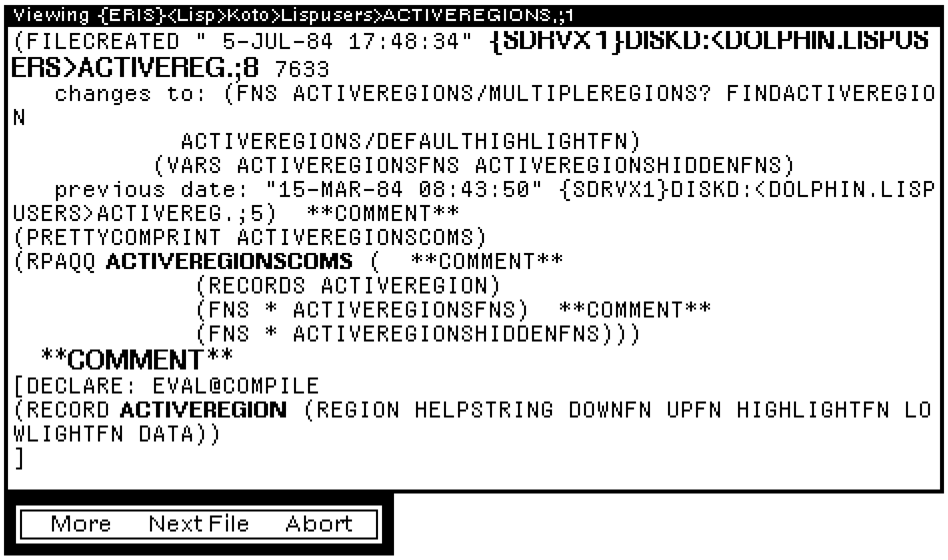
SEE

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" {SDRVX1}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.

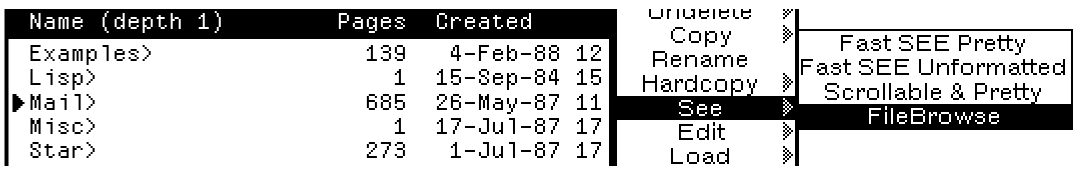
The unformatted version of the SEE command does no special processing on the characters whatsoever. It simply displays each eight-bit byte as a single character, uninterpreted. This means that bytes that do not represent normal printing characters may be displayed as black boxes, in the form ^x or #x, 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 ">".

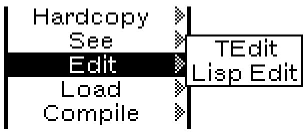


```
 Name (depth 1)          Pages  Created
 Examples>                 139   4-Feb-88 12
 Lisp>                       1  15-Sep-84 15
▶Mail>                     685  26-May-87 11
 Misc>                       1  17-Jul-87 17
 Star>                     273   1-Jul-87 17
```

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.

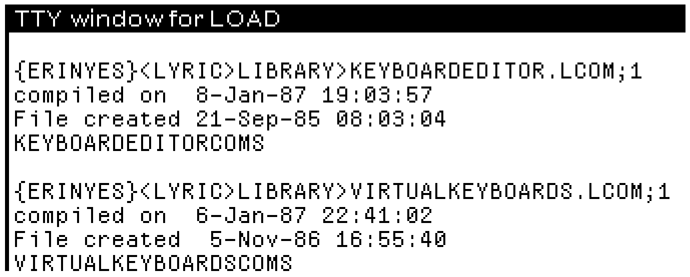
The subcommand LISP EDIT is appropriate for Lisp source files produced by the file manager. It calls the Lisp structure editor on the file's coms. If the file is not yet known to the file manager, you will be asked whether you want to load it first (using LOAD PROP). If not, the operation is aborted.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

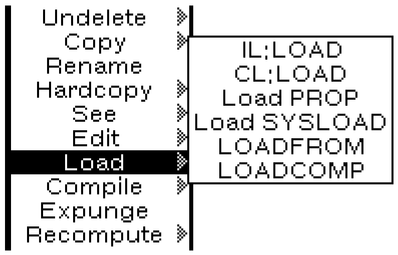
FileBrowser's LOAD command can be used to load both source (interpreted) and compiled files into your workstation's virtual memory. First select the file or files you want to load, then select Load with the left mouse button.

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.

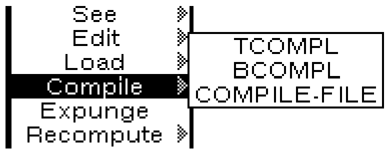
LOADFROM calls the file manager's function LOADFROM. This loads variables and other expressions but not Interlisp functions, and does so in a way that informs the file manager, so that the editor knows where to find the functions.

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

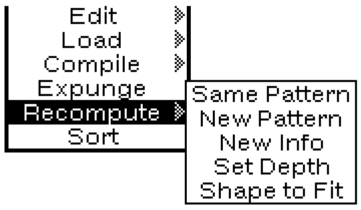
FileBrowser's display shows those files that existed and matched the specified pattern at the time you created the browser. If you want the browser to reflect the latest state of the file system, use the RECOMPUTE command.

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.

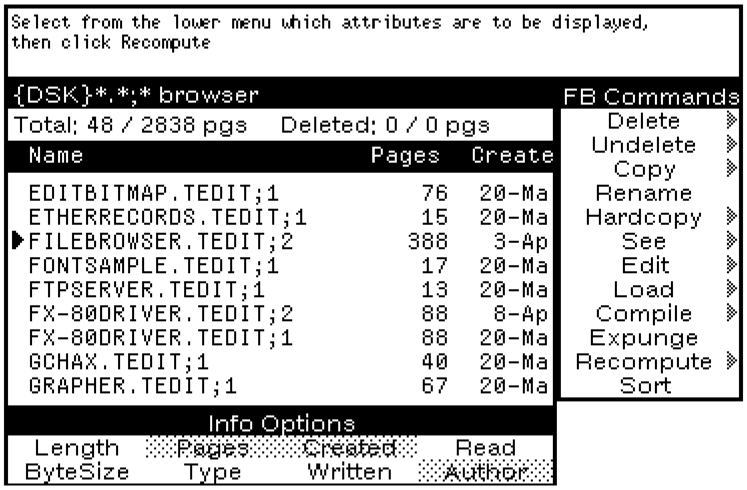
SET DEPTH lets you change the depth to which FileBrowser enumerates a directory on an XNS file server. It is described in the section "SET DEPTH".

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                                FB Commands
Total: 48 / 2838 pgs    Deleted: 0 / 0 pgs         Delete    >
  Name                            Pages  Create    Undelete  >
                                                   Copy      >
  EDITBITMAP.TEDIT;1                76   20-Ma     Rename
  ETHERRECORDS.TEDIT;1              15   20-Ma     Hardcopy  >
▶FILEBROWSER.TEDIT;2              388    3-Ap     See       >
  FONTSAMPLE.TEDIT;1                17   20-Ma     Edit      >
  FTPSERVER.TEDIT;1                 13   20-Ma     Load      >
  FX-80DRIVER.TEDIT;2               88    8-Ap     Compile   >
  FX-80DRIVER.TEDIT;1               88   20-Ma     Expunge
  GCHAX.TEDIT;1                     40   20-Ma     Recompute >
  GRAPHER.TEDIT;1                   67   20-Ma     Sort

                  Info Options
  Length     Pages     Created     Read
  ByteSize   Type      Written     Author
```

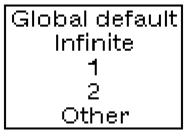
The Info Options items have the following meanings:

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 edior 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
>
> *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`.

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]

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)`.

[This page intentionally left blank]

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 does not 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
```

[This page intentionally left blank]

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.

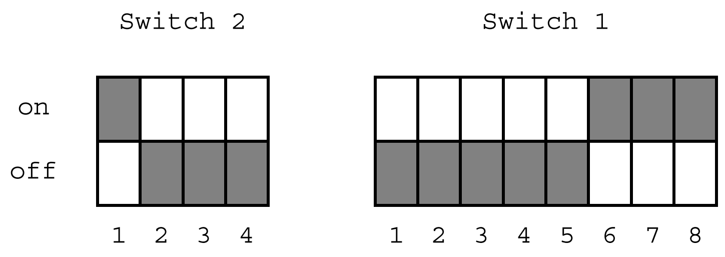


Figure 2. FX-80 DIP switch settings

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 multiple 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, bitmaps, 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 the Limitations section 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 GRAPHNODE Record section), 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*.

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 `TONODE`s of a node are positioned beneath it, and the `FROMNODE`s 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. `NIL`s cause “pretty” defaults based on the size of *FONT*.

PERSONALD Is specified in points; it controls the minimum distance between any two nodes.

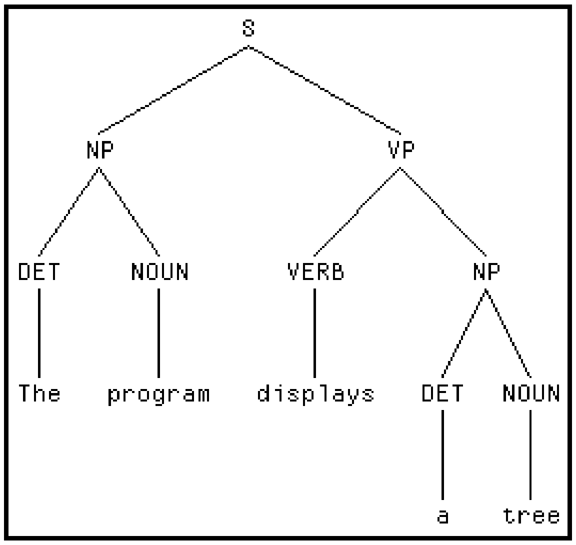
MOTHERD Is the minimum distance between a mother and her daughters.

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 Performing Other Tasks section), 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 the following:

```
(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)))
```



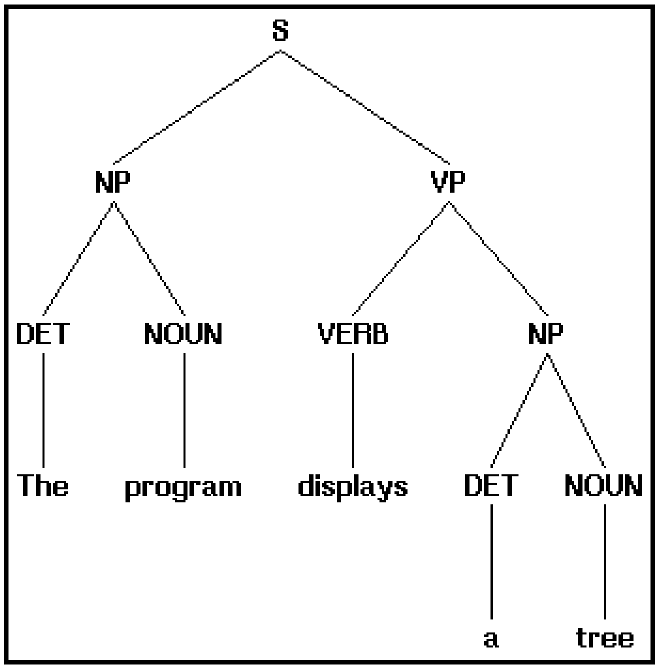
(LAYOUTSEXPR *SEXPR FORMAT BOXING FONT MOTHERD PERSONALD FAMILYD*) [Function]

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 on which the graph is displayed. *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.

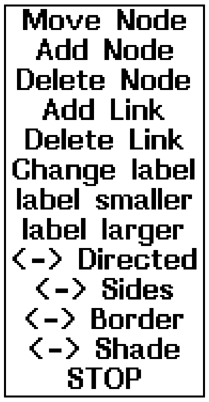
The initial value of GRAPH/HARDCOPY/FORMAT is set to
(MODE PORTRAIT PAGENUMBERS T TRANS NIL)

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.

Even if you do not specify any of the following fields (except GRAPH.ADDNODEFN), the system provides a set of functions which allows you to edit the graph. Any functions you specify are executed *after* the default function is executed. For example, if you supply the ADDLINKFN and then add a link using the edit menu, your function is called after the link is added. Exception: for ADDNODEFN, the function is called and must return a node to be added to the graph; this function is executed *instead* of the default function.

`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 the following 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 is 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 sits completely above and to the left of the reference point, and one means it sits 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. `NODEID`s are used in the link fields instead of pointers to the nodes themselves, so that circular Lisp structures can be avoided. `NODEID`s are often used as pointers to the structure represented by the graph.
`TONODE`	Is a list of `NODEID`s. A link runs from the currently selected node to each node in `TONODE`s. Entries in this field can be used to specify properties of the lines drawn between nodes.

If an item in the `TONODE`s 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 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 bitmap. 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 has a menu above it that contains the items LARGER and SMALLER. Selecting in the menu with the left button causes a small change in scale; selecting with the middle button causes a larger change in scale.

The graph zooms 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 are printed in the closest Helvetica size. This is often smaller than the corresponding Gacha font.

[This page intentionally left blank]

HASH

Note: This module is provided for backwards compatibility. New applications should use the HASH-FILE Library Module instead of this module.

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 arbitary Lisp expressions. The associates 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, a 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 it 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 hashfile with this module, you have to open it using the following 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 the error "not a hashfile" if *FILE* is not recognized as a hash file.

ITEM LENGTH 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.

(GETHASHVILE *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) deletes a key and returns 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 (through 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 prints 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 need a custom *COPYFN* (supplied in the call to `CREATEHASHFILE`), which is used to copy data during the rehasing 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 is returned is used as the value of the key in the new hash file. (This lets you intervene, perhaps to copy out-of-bank data associated with *VALUE*.)

VALUETYPE is a no-op.

If *LEAVEOPEN* is non-`NIL`, the new hash file datum is returned open. Otherwise, the new hash file is closed and the name is returned.

(`HASHFILEPLST` *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`.

`HASHFILEDTBL` [Variable]

The hash file read table. Default is `ORIG`.

`HASHLOADFACTOR` [Variable]

The ration, used slots/total slots, at which the system rehashes the file. 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 milion bytes. No range checking is done for these limits.

Two-key files operate on pnames only, without regard to packages.

[This page intentionally left blank]

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 are 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 grows 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 `:in` put then storing values in the hash file is 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 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 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.

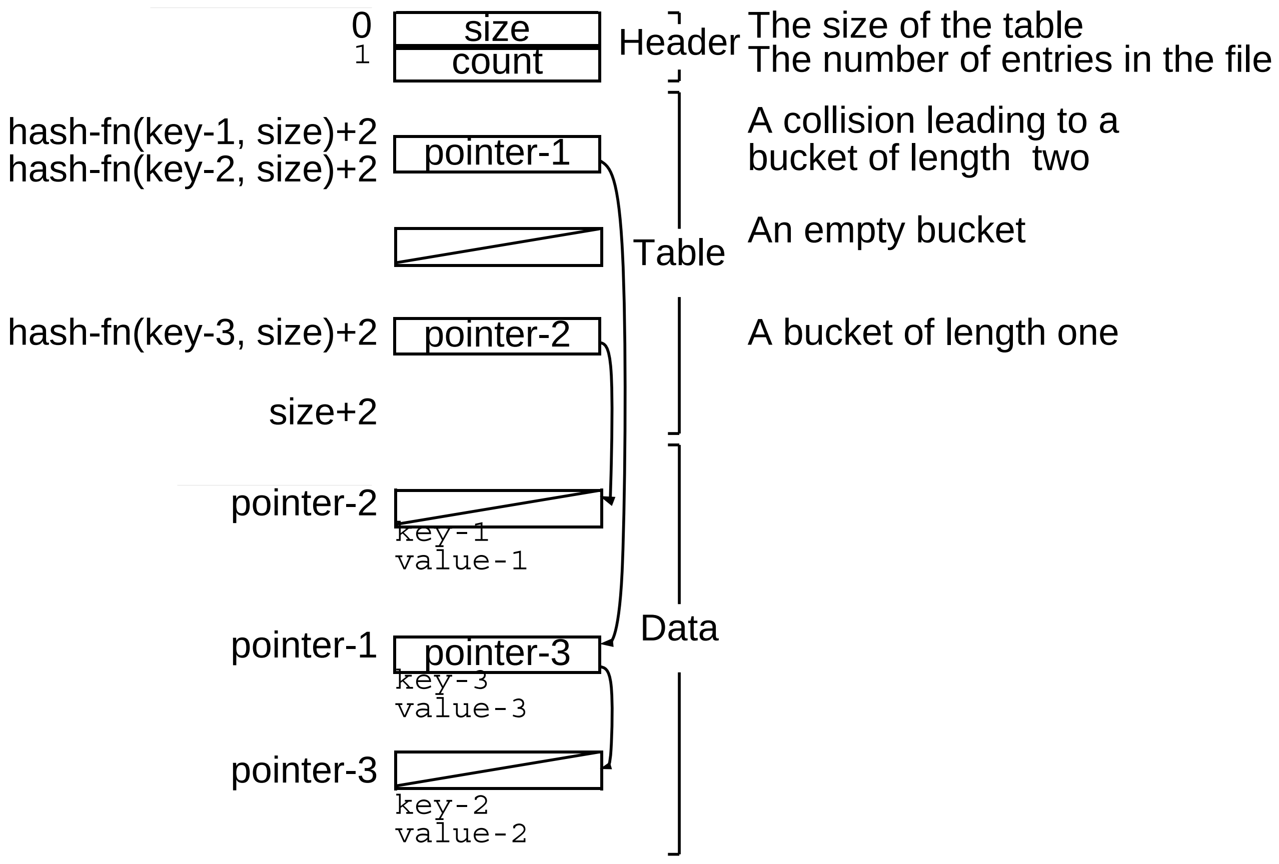


Figure 3. Hash File Format

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 is 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 is 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 is 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 is 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

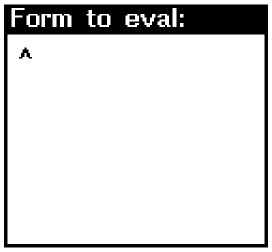
Installation

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 closes, and a rule of the specified size is 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 2½-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 may 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

Theoretically, a rule can be 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 usually cannot tell the difference between rules that are less than ½ 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.

½ point rule

1-point rule

1½-point rule

2-point rule

2½-point rule

3-point rule

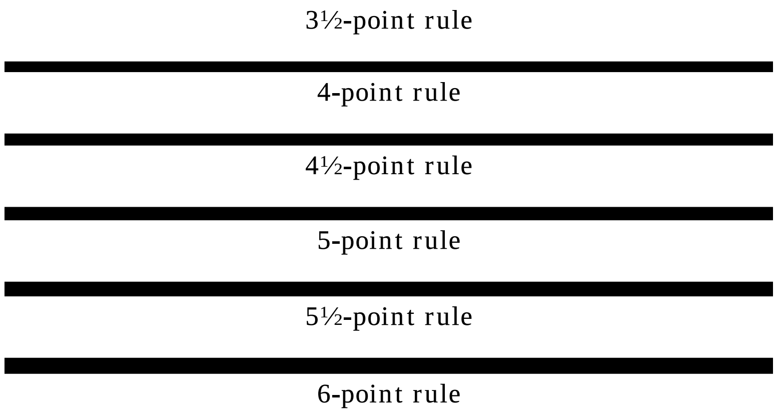
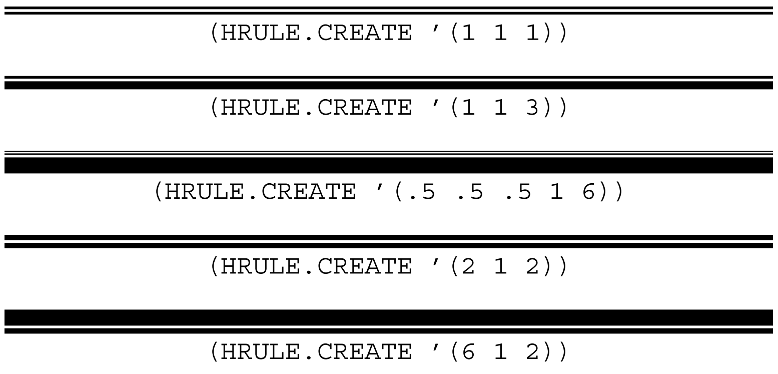


Figure 4. Horizontal rules

Shown in Figure 5 are some examples of built-up rules, along with what you would type to create them:



```
(HRULE.CREATE '(1 1 1))
(HRULE.CREATE '(1 1 3))
(HRULE.CREATE '(.5 .5 .5 1 6))
(HRULE.CREATE '(2 1 2))
(HRULE.CREATE '(6 1 2))
```

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 to run this module successfully:

- `KERMIT, KERMITMENU`
- `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 takes place. If *WINDOW* is `NIL`, the value of `CHATWINDOW` (the first Chat window to be opened) is 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 takes place. If *WINDOW* is `NIL`, the value of `CHATWINDOW` (the first Chat window to be opened) is 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 is blank, and cumulative packet counts and other messages are 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 takes 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.

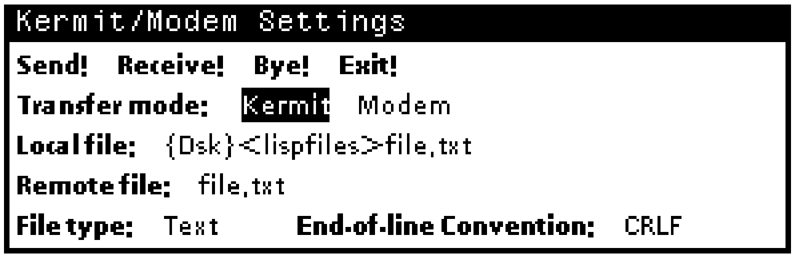
WINDOW is the Chat window over which the transfer takes place.

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 contains an entry labeled "Kermit" near its top. If you select this entry, a Kermit menu appears 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	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	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	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	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`	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

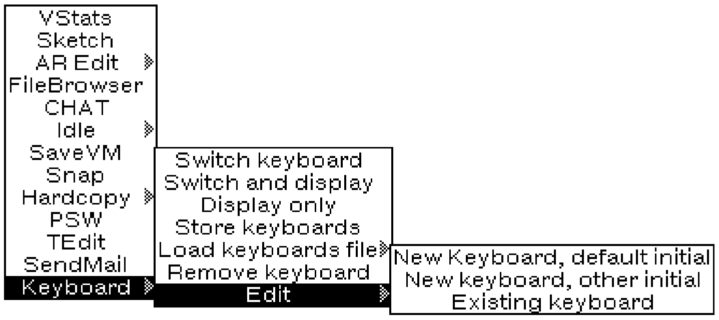
Load `KEYBOARDEDITOR.LCOM` and `VIRTUALKEYBOARDS.LCOM` from the library.

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`.

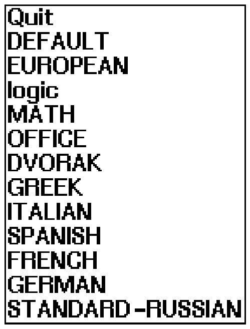


Creating a New Keyboard From a Copy of the Default Keyboard

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 prompts 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 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 are prompted for a name for the new keyboard. The system then displays a menu of the known keyboards from which to choose 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 are not 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 is done on that keyboard and the second argument is ignored.

If *KEYBOARD* is a new name, the editing is done on a copy of *INITIALKEYBOARD*, with *KEYBOARD* as its new name. If *INITIALKEYBOARD* is NIL, the default keyboard is used as a base keyboard.

Examples:

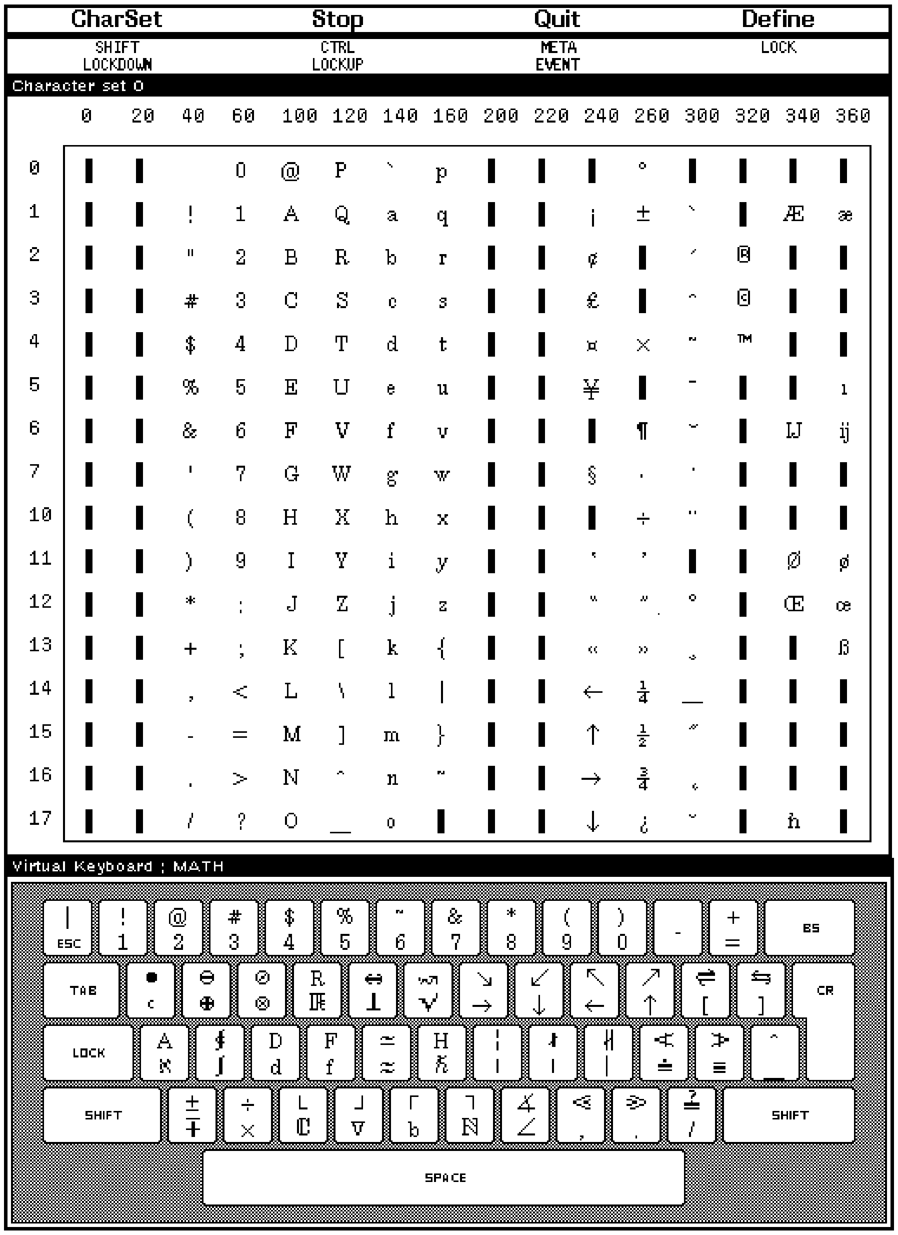
To create a totally new virtual keyboard, call `(EDITKEYBOARD` *NEWNAME*`)`.

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, the command menu is at the top, the character menu in the middle, and the keys menu at the bottom.



	0	20	40	60	100	120	140	160	200	220	240	260	300	320	340	360
0	▌	▌		0	@	P	`	p	▌	▌	▌	°	▌	▌	▌	▌
1	▌	▌	!	1	A	Q	a	q	▌	▌	¡	±	`	▌	Æ	æ
2	▌	▌	"	2	B	R	b	r	▌	▌	¢	▌	´	®	▌	▌
3	▌	▌	#	3	C	S	c	s	▌	▌	£	▌	^	©	▌	▌
4	▌	▌	$	4	D	T	d	t	▌	▌	¤	×	~	™	▌	▌
5	▌	▌	%	5	E	U	e	u	▌	▌	¥	▌	¯	▌	▌	ı
6	▌	▌	&	6	F	V	f	v	▌	▌	▌	¶	˘	▌	Ĳ	ĳ
7	▌	▌	'	7	G	W	g	w	▌	▌	§	·	˙	▌	▌	▌
10	▌	▌	(	8	H	X	h	x	▌	▌	▌	÷	¨	▌	▌	▌
11	▌	▌	)	9	I	Y	i	y	▌	▌	‘	’	▌	▌	Ø	ø
12	▌	▌	*	:	J	Z	j	z	▌	▌	“	”	˚	▌	Œ	œ
13	▌	▌	+	;	K	[	k	{	▌	▌	«	»	¸	▌	▌	ß
14	▌	▌	,	<	L	\	l	|	▌	▌	←	¼	_	▌	▌	▌
15	▌	▌	-	=	M	]	m	}	▌	▌	↑	½	˝	▌	▌	▌
16	▌	▌	.	>	N	^	n	~	▌	▌	→	¾	˛	▌	▌	▌
17	▌	▌	/	?	O	_	o	▌	▌	▌	↓	¿	ˇ	▌	ħ	▌

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 pops 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.

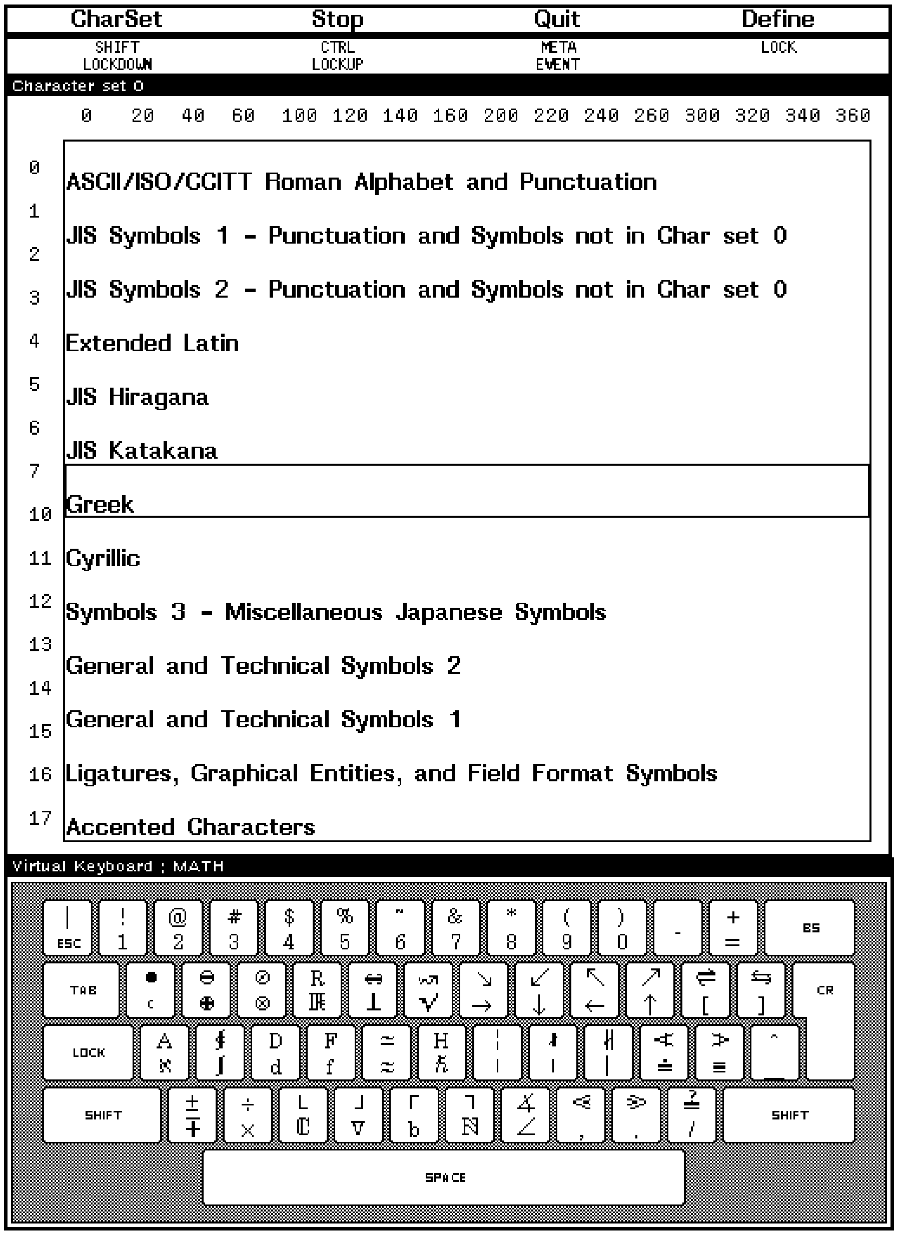

CharSet
Stop
Quit
Define
SHIFT
LOCKDOWN
CTRL
LOCKUP
META
EVENT
LOCK
Character set 0
0 20 40 60 100 120 140 160 200 220 240 260 300 320 340 360
0 ASCII/ISO/CCITT Roman Alphabet and Punctuation
1 JIS Symbols 1 - Punctuation and Symbols not in Char set 0
2 JIS Symbols 2 - Punctuation and Symbols not in Char set 0
3 Extended Latin
4 JIS Hiragana
5 JIS Katakana
6 Greek
7 Cyrillic
10 Symbols 3 - Miscellaneous Japanese Symbols
11 General and Technical Symbols 2
12 General and Technical Symbols 1
13 Ligatures, Graphical Entities, and Field Format Symbols
14 Accented Characters
Virtual Keyboard ; MATH
ESC
TAB
LOCK
SHIFT
SHIFT
BS
CR
SPACE

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 ise 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 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 sends 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 sends the shifted character to the current input stream.

If a key has the LOCKSHIFT property, the lock key is inverted in the keys menu. To change the LOCKSHIFT property of a key, first make the shifted key the current key. 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 adds the newly created keyboard to the list of known keyboards (it will thus appear on future menus). `Quit` exits after modifying the virtual keyboard; `Stop` exits without modifying the keyboard. In both cases the new keyboard is 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]

An Alist of the IDs you use for the keys in the rest of the configuration; i.e., your names for the keys. For simplicity, these are usualy 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 `KEYACTION`s (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 do not 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.

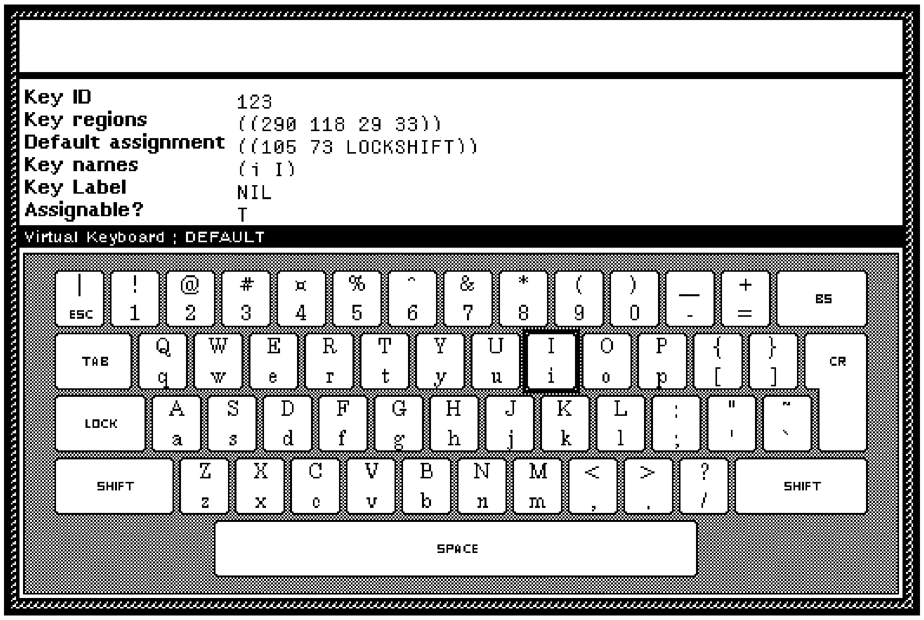
Note: You must save your own configurations. There is no user interface for saving them, nor any automatic scheme.

Editing a Keyboard Configuration

Once you have created a KEYBOARDCONFIGURATION, you can make modest changes to it using the function:

(EDITCONFIGURATION *CONFIGNAME*) [Function]

where C*ONFIGNAME* is the CONFIGURATIONNAME you have assigned to your new configuration. This creates a virtual keyboard editing 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 Editing 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 are 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]

Lafite Changes

```
Folder names -> strings
Profile handling (LAFITE.PROFILE.VARS, "Lafite.info")
Binding *UPPER-CASE-FILE-NAMES*
Lafite modes all active.  LAFITE.USE.ALL.MODES = T or list of operations from {:GETMAIL, :POLL}
Mail retrieved only from servers that claim to have new mail as of most recent poll.
Handling NS chars, eol conventions in parser
BADTOCFILE returns nil so don't get bogus "Folder is Empty"
LAFITE.BROWSER.LAYOUTS, LAFITE.DISPLAY.SIZE
\LAFITE.BROWSE no longer takes file & options
new fn LAFITE.BROWSE.FOLDER
toc handles ns strings
LAFITE folder arg can be list of (folder region displayregion iconpos . options)
LAFITE options can include :ACTIVE, :GETMAIL, :SHRINK
Browser menu labels narrower
LAFITE.FOLDER.MENU.FONT
LAFITE.MIDDLE.UPDATE = ({:shrink|:close} {:update|:expunge})
LAFITE.BROWSER.ICON.PREFERENCE is NIL (default), :ASK (prompt for position) or a fn.
Recache command
LAFITE.DONT.DISPLAY.HEADERS, LAFITE.DONT.FORWARD.HEADERS, LAFITE.DONT.HARDCOPY.HEADERS
Fixed hardcopy toc
Message display titlebar menu [LAFITE.EXTRA.DISPLAY.COMMANDS, LAFITE.LOOKS.SUBCOMMANDS]
Middle-button display windows stay up
Middle-button browser icon offers to GetMail if "active" (has had a getmail done to it or was the
        active file browsed when Lafite was turned on).
LAFITE.SEND.FORMATTED = NIL|:ASK|T or list of (type answer).
Can now send plain text msg with ns chars
NS mail handles attachments, including references.
NS mail handles TransportProblem.
Copy selection in browser.
Browse & Forget.
MOVETOCONFIRMFLG only affects menu selections.
EOM works again in LispCore
LAFITE.SIGNATURE, LAFITE.GV.FROM.FIELD & handling of same when username changes.
*GV-SHOW-POSTMARK* = T retains GV information.
```

Deleted or renamed fns/vars:

```
LA.REMOVEDUPLICATES
LA.SETDIFFERENCE
PROFILEFILENAME
\LAFITE.MERGE.PROFILES -> \LAFITE.MERGE.NAMELISTS
LAFITEDISPLAYREGION
LAFITEIMMEDIATECHANGESFLG, CHANGEFLAGINFOLDER
MSGFOLDERTEMPLATE, MSGFOLDERICON, MSGFOLDERMASK -> LAFITE.FOLDER.ICON
MSGUNSENTICON, MSGUNSENTMASK, MSGUNSENTREGION -> LAFITE.MSG.ICON
use of FOLDERDISPLAYWINDOWS field, now uses FOLDERDISPLAYREGIONS (replace BROWSERSELECTIONREGION)
\LAFITE.BROWSE family completely changed
\LAFITE.MOVETO[.PROC] changed
\LAFITE.GETMAILFOLDER deleted.
```

Lafite
The Interlisp Mail System

Documentation: {Eris}<Lisp>Library>Lafite.Press
Program: {Eris}<Lisp>Library>Lafite.Dcom
Revised: February 22, 1984 by Bill van Melle

General comments

Lafite is the Interlisp system for reading and sending mail. Lafite retrieves inbound mail from the user's *inboxes* on one or more mail servers. Mail is retrieved into one or more *mail folders*, which can be any Interlisp-accessible file. One can open a *browser* on a mail folder, which allows the folder's contents to be displayed, deleted, moved into other folders, hardcopied, etc. Messages can be composed using the standard Interlisp text editor and all its facilities, then sent to other users.

Lafite permits communication with various sorts of mail servers. Currently implemented are interfaces to the Grapevine mail system and the NS mail system.

Mail systems are notorious for inspiring lengthy wish lists for new functionality or user interface embellishments. Feel free to send your suggestions to LafiteSupport (using Lafite's "Lafite Report" form, described below).

Compatibility with Laurel

Previous users of the Laurel mail program will find Lafite to have a similar user interface. Some of the ways in which it differs from Laurel are the following:

> Laurel can only access mail folders on the local disk; Lafite can access folders on remote file servers. Thus, it is not necessary to transfer mail folders back and forth between your local disk and your file servers.
>
> Laurel can only "browse" one mail folder at a time; Lafite can have several mail folders "opened" at the same time. Utilizing the Interlisp window system, you can view the table of contents of many mail folders and refer to messages in them independently.
>
> You may have multiple windows displaying messages and multiple windows for sending messages and you may move text freely among them.

Lafite can read mail files written by the Laurel and Hardy mail programs; the files it writes are in Laurel format.

Loading Lafite

Lafite consists of two parts: a generic part that manages browsers, mail folders and message composition; and a protocol-specific part that manages communication between Lafite and the remote mail system. You have to load both parts. The first is the file `LAFITE.DCOM`, plus the files that it automatically loads. The second depends on your network environment. For sites that use the Xerox NS Mail system, you should load the file `NSMAIL.DCOM`. For Grapevine sites, load the file `MAILCLIENT.DCOM`. If your site has a choice of mail server types, you can load all the relevant protocol files and then choose among them by setting the mode (see **Lafite Modes**, below).

Lafite operation

When Lafite is loaded, it can be started by calling `(LAFITE 'ON` *MAILFOLDER*`)`. Lafite will attempt to establish the user's mail server identity, read stored user data from the file `LAFITE.PROFILE` (if it exists) and bring up the Lafite Status window. It will then establish a browser for the file *MAILFOLDER*, creating an empty mail folder of that name if one does not exist. If *MAILFOLDER* is not supplied then `ACTIVE.MAIL` will be used (see `DEFAULTMAILFOLDERNAME` below). If *MAILFOLDER* is supplied but is the atom `NIL`, then no mail folder is opened but you are free to send mail and open any mail folder at a later time.

Mail files must reside on randomly-accessible devices. In the current Interlisp environment, this means they must reside on `{DSK}`, `{FLOPPY}`, or a file server that supports the Leaf protocol.

There are three major types of windows used by Lafite: the Status Window; Browser Windows, which are views on particular mail folders; and Message Composition Windows. Each type of window has its own fixed menu of commands. In general, while a command is "in progress", the menu item that invoked it is greyed out. Commands may be selected with the LEFT or MIDDLE mouse buttons; in some cases, the MIDDLE button provides some sort of special treatment, described in the documentation of such commands.

The Lafite Status Window

The Lafite Status window contains a small, fixed menu and a region for Lafite status information. While Lafite is in operation, it runs a background process—`LAFITEMAILWATCH`—that polls the user's inboxes periodically and reports in the status region if there is new mail. Clicking in the status region causes the background process to wake up and report status immediately, instead of waiting its normal interval. The commands in this window's menu are as follows:

Browse — pops up a menu of the mail folders that Lafite is aware that you have. Selecting 'Another Folder' prompts you for a folder name in the prompt window. After a folder is selected, a *browser* window onto that mail folder is opened.

Selecting the **Browse** command with the MIDDLE button brings up a menu of Browse-related commands. In addition to simple **Browse** there are these commands:

Browse Laurel Folder — Browses a file that was produced by the Laurel mail reader version 6.1 or later. Laurel 6.1 files are almost the same as Lafite files, but contain some line-formatting information that is stripped out by this command. After you have applied this command once to a file, you can subsequently browse the file with the normal **Browse** command (unless you use Laurel on it again, of course). **Important Note**: this command is not intended for repeated use, but simply to make mail files built by Laurel more pleasing to browse. This command has the side effect of destroying any Tedit formatted messages, so should be used with care.

Forget Folder — Removes a folder from the list of known mail folders.

Forget Message Form — Removes a message form from the list of known message forms (see **Save Form** command).

Send Mail — brings up an Interlisp text editor window on a message form. The form is a canonical "empty message" if **Send Mail** is selected with the LEFT button. If **Send Mail** is selected with the MIDDLE button, a menu is presented with the following choices:

Standard Form—provides an empty message template (same as using the LEFT button).

Lisp Report—provides a message template to report an Interlisp bug or make a suggestion.

Lafite Report—provides a message template similar to **Lisp Report** but sent to Lafite maintainers.

Saved Form—prompts for a form name, which can be any text file, or a form created by the **Save Form** command (below).

Also in the menu are any names of known user-defined message forms created by the **Save Form** command. Each message form runs in its own process, so you can have several in progress at once. When you have finished composing the message, click **Deliver** in the message's menu.

Quit — Stops Lafite and closes all browser windows. If any of the associated mail folders need updating, prompts with a menu asking what degree of updating should be performed (see **Update** command). It also saves the names of known mail folders and form files on the file `LAFITE.PROFILE` so that this information will be available when you next run Lafite. You may achieve the same result under program control by calling `(LAFITE 'OFF)`, rather than buttoning this menu item. Most users find that they never invoke 'Quit', but rather keep Lafite always active in the background.

Selecting the **Quit** command with the MIDDLE button brings up a menu of status changing commands. This menu includes items for changing Lafite's mode (see **Lafite Mode**), and the command **Restart**, which is equivalent to `(LAFITE 'OFF)` followed by `(LAFITE 'ON NIL)`.

Browser Windows

A Browser window is a view onto a mail folder. The main part of the window displays the table of contents, a one-line summary of each message. This window is scrollable in both dimensions. Above the table of contents is a horizontal menu containing commands specific to the mail folder associated with the browser window. Above the menu is a prompt window, in which various status information related to the browser is printed, and where some information is prompted for.

Browser comands operate on the currently selected set of messages. A selected message is indicated by a black triangle to the left of its message number. A message can be selected by clicking anywhere inside its summary line. The type of selection depends on which button is used:

left button — selects just this single message, deselecting any other selected message.

middle button — adds a message to the current selection.

right button — extends the selection up or down. Deleted messages are not included in this extension unless the control key (CTRL) is down.

shift key (SHIFT) and any button — removes this message from the current selection.

Laurel users note: messages can be selected by clicking *anywhere* within the summary line (except for the mark area, see below), unlike in Laurel, where you must select at the left end.

The commands in the browser menu are as follows:

Display — displays the selected message. If the selected message is already displayed, the selection is advanced to the next undeleted message, and this message is displayed. If there is more than one message selected, buttoning **Display** cycles through the messages.

If you select **Display** with the LEFT button, the message is displayed in the primary message display window for the browser, replacing any previously displayed message. If you select **Display** with the MIDDLE button, the message is displayed in a newly created window, for which you will be prompted. Using the MIDDLE button you can make multiple windows containing messages for further reference (e.g., to use in composing your own message).

Delete — deletes the selected messages. A deleted message is indicated by a black line through its summary line. The message is not actually removed from the mail folder until you Expunge (see Update, below).

Undelete — undeletes the selected messages.

Answer — constructs a Message Composition Window containing an *answer template* for the current message. After you deliver the answer, an "**a**'' will appear in the browser window as the message's mark.

Forward — similar to 'Answer' but the message form is a *forward template* for the currently selected messages. After you deliver the forwarded the messages, an "**f**'' will appear as the message's mark.

Hardcopy — prints the selected messages on your local printing device. When the hardcopy is complete, the message's mark is changed to "**h**'' if the message didn't already have a more interesting mark. Messages can be marked for hardcopy, but the actual printing deferred until later; see description of `LAFITEHARDCOPYBATCHFLG`. Currently, one should exercise care in hardcopying large sets of messages, as Lafite makes no attempt to perform the hardcopying in smaller pieces; too large a body of messages can make printers unhappy, or exhaust local disk space in the process.

Move To — pops up a menu of known mail folders and moves the selected messages to the chosen folder. A new mail folder can be created by selecting 'Another Folder' and typing in the mail folder name in the prompt window. You will then be asked to confirm the move. When the move is successful, the messages are marked deleted in the source browser window, and given the "**m**'' mark.

The name of the mail folder you most recently moved messages to appears in the title bar of the browser window as the "Default 'Move To':" folder. You can "accelerate" subsequent Move operations by selecting the 'Move To' command with the MIDDLE button. This will perform the move to the Default 'Move To' folder without bringing up a menu.

Update — The changes that you make to your mail folder (deletions, changes of message marks, etc) are not actually transmitted to the physical mail file until you perform the Update command. There are two subcommand choices for Update:

> **Write out Changes Only** — makes the browser and the mail file completely consistent with one another: if you were at this point to logout from Lisp, run Lafite in another incarnation of Lisp, and browse the same mail folder, you would get a browser that was in exactly the same state, deleted messages and all. This command involves writing out to your mail file and its table of contents the information about what has changed: new marks, deletions, newly parsed messages.
>
> **Expunge Deleted Messages** — in addition to making the browser and the mail file completely consistent with one another, this command compacts your physical mail file so as to remove all messages marked deleted.

Which to choose? You eventually want to Expunge, so as to reduce the amount of file space your mail requires, but Expunge is not always fast. **Expunge** requires rewriting your mail file starting at the first deleted message, so is somehow "proportional" to the number of undeleted messages beyond that point. **Write out Changes Only** is proportional to the number of changes, and is thus usually faster than **Expunge**, except when the changes consist primarily of a string of deleted messages at the end of the folder.

If you Close or Shrink a Browser window that has had changes to it, you are prompted with a menu offering to **Write out Changes**, **Expunge**, or make no change before the window is closed/shrunk.

If you have deferred hardcopy, the Update menu also includes a choice **Do Hardcopy Only** if you want to print out your messages but not actually update the file.

Get Mail — brings new mail into the folder that this browser is viewing.

Changing the Message Mark

Each message in a mail folder has a ”message mark”, which is an additional tidbit of information about the message displayed in the summary line in the browser. Messages originally come in with the mark ”**?**”, meaning they are unexamined. Some Lafite commands change the mark automatically. For example, displaying a message changes its mark from ”**?**” to a blank; answering a message changes its mark to ”**a**’’. You can change the mark directly by selecting with the mouse in the narrow area immediately to the left of the message number, where the mark is printed. Simply click in the mark position, and type the new mark (a single character).

The Table of Contents

To speed the browse operation, Lafite maintains, for each mail folder you browse, a ”table of contents” file, which contains all the information displayed in the browser window. This file is named by concatenating the name of the mail file with ”-LAFITE-TOC’’. This file is completely redundant, in that if it doesn’t exist, Lafite simply recomputes it by parsing the entire mail file. Lafite has some rudimentary checks to guarantee the consistency of the contents file with the mail file it describes; Lafite deletes and recomputes the contents if it suspects that the table of contents file is out of date.

Message Composition Windows

On top of the text editor window is a horizontal menu for telling Lafite what to do with the message being created in the text editor window. When you have transformed the text to the desired message, select one of the following menu items:

Deliver — sends your message. This process happens in background, so you can proceed with anything else you desire while delivery proceeds. If successful, the window closes and an entry is made in your *outbox* (see below). If the delivery fails, for any of a variety of reasons (e.g. bad address fields, the mail server timed out) which will be displayed in the prompt window, the message is redisplayed and you are back in the text editor to re-edit and then resend the message. During the delivery process, the menu atop the message window changes into a single item, **Abort**; if you click this item, the delivery is aborted, and you are returned to editing your message.

Save Form — asks you for a file name on which to save this message for later use. This is the way to save a message form for later repetitive use. It does *not* send it. If no extension is given for the file name, it defaults to the value of LAFITEFORM.EXT (initially LAFITE-FORM).

Of course, you can always bring up the text editor command menu by MIDDLE buttoning the title bar of the editor window. You can then do any text editor command. If you select ’Quit’ you will immmediately leave the text editor, bypassing the above commands. You can also simply close a window using the normal window command menu if you decide you don’t want to do anything with a message you started composing.

When editing message forms, fields that should be filled in are enclosed in ”>>” and ”<<”; e.g., >>Recipients<<, >>Subject<<. To make it easier to fill these in, the first field is highlighted in delete mode. Each successive field can be reached by typing the middle-blank key on the keyboard (or OPEN on the Dandelion keyboard, or whichever key you have assigned the TEdit ”Next” syntax to). See the TEdit documentation for details.

In a saved form, if you insert a field ”>>Self<<”, it will be automatically filled with the name of the currently logged in user when retrieved into a message composition window. This facilitates making forms that many different people can use. Of course, if you edit an already saved form containing ”>>Self<<” in a message composition window, you will have to perform the inverse, replacing your name with ”>>Self<<”, before saving it away again.

After a message is succesfully delivered, it is entered into your *outbox*, a window attached to the bottom of your status window. This window contains a one line description of each of the most recent *n* messages you have sent, where *n* is the value of the variable `LAFITEOUTBOXSIZE`, initially 2. The outbox is treated as a menu—selecting a line in it brings up the corresponding message for further editing and delivery. The outbox can be independently closed, if you are no longer interested in the messages displayed therein and want to free up the resources that they are tying down.

Format of Message Headers

The header of a message contains a number of fields, each on a separate line, followed by a blank line to separate the header from the message body. Each header line consists of a field name followed by a colon, then the contents of the field. Messages standardly have "To:" and "cc:" fields consisting of one or more recipient names, separated by commas (and spaces if desired). Lafite automatically supplies a "From" field containing your name. If you want the message to appear to be "from" someone else (e.g., if you are sending the message from someone else's logged in Lafite), or from more than one user, you can supply your own "From" field. In this case, Lafite will supply a "Sender" field to show who actually sent the message.

Sending Formatted Messages

You have available to you the full power of TEdit when you are composing a message. This means you can change fonts, format paragraphs, and insert "image objects". If you try to deliver such a formatted message, Lafite will ask if you want to retain the formatting information, putting up a menu with these choices:

Send Formatted Message — retains all the formatting information. Note that only Lafite users can read formatted messages; all other mail readers will see the plain text of the message. Even if all of the recipients are Lafite users, keep in mind that retaining the formatting information means in particular retaining the fonts you used in composing the message; not everyone likes to read mail in the font you chose, so if the only "formatting" is a nonstandard font that isn't important to the appearance of the message, do not make this choice.

This choice is made automatically if the message contains image objects, as there is no way to send the images without the formatting.

Send Plain Text — sends only the text of the message. The message will appear to the recipient in whatever font her mail reader standardly uses, and all paragraph formatting (centering, justification, special tabstops) will vanish.

Abort — does not send the message, but returns to the message editor to allow you to continue editing the message.

Lafite Modes

Lafite is capable of sending and retrieving mail via different protocols. At any one time, however, it only operates with a single protocol set, which is considered its current "mode". The currently implemented modes are GV (Grapevine) and NS. The easiest way to change the mode is to use the MIDDLE-button menu under **Quit**. Lafite's mode can also be changed programmatically with the function `LAFITEMODE`:

`(LAFITEMODE` *MODE*`)`

Returns the current mode, a litatom. In addition, if *MODE* is non-`NIL`, changes Lafite's mode to be *MODE*.

When Lafite is first turned on, it chooses its mode as follows: If there is only one mode supported (i.e., only the files supporting one protocol have been loaded), it chooses that mode; otherwise, if `LAFITEMODEDEFAULT` is non-`NIL`, it chooses that mode; otherwise, the user must set the mode manually. The current mode is displayed in the status window if LAFITESHOWMODEFLG is true and more than one set of mail protocol implementations has been loaded.

You can freely intermix mail of the various modes in one folder, but the current implementation is not very clever about it. For example, the Answer command always treats the selected message as if it were one in the current mode. So if you try to answer a Grapevine message while in NS mode, some confusion may result.

Grapevine

The Grapevine implementation of mail protocols is obtained by loading the library file MAILCLIENT.DCOM.

Grapevine addresses are of the form "name.registry", e.g., "Carstairs.pa". If you omit the registry, it is defaulted to the value of DEFAULTREGISTRY. Arpanet recipients are of the form "name@host". In addition, the following forms of address are recognized, where "actual address" must be a valid Grapevine or Arpanet address:

Human sensible name <actual address>

actual address (random comments)

"Comments, including parentheses and commas" <actual address>

If you address a message to a public distribution list (a Grapevine name ending in the character "↑") and have not included a Reply-to field, Lafite will prompt you to supply one. Your choices are to include a Reply-to: self field, so that answers to this message will be sent only to you; a Reply-to: other field (you get to fill it in yourself), or no Reply-to: field at all. It is important to have a Reply-to field in messages sent to distribution lists, so that casual users will not inadvertantly reply to the entire distribution list when a reply to you only was intended.

Please be careful in using public distribution lists. Keep in mind that your one message will be received by many people—is what you have to say important enough to be worth taking their time? There are certain organizational distribution lists, e.g., AllPA↑.pa, or ISL↑.pa, that you should definitely avoid sending casual messages to, as they are large and their members are not voluntary. Other distribution lists are really "interest lists" whose members have voluntarily consented to receive messages in the general category of movies, concert announcements, humorous anecdotes, or whatever. Messages to these lists are less constrained, but you should still take a moment to think about whether your message is appropriate. I highly recommend that all electronic mail users read Chapter 6 of the Laurel manual, "Message system mores".

Shortcomings as of this writing: Private distribution lists are not yet supported. And the Answer command does not preserve the entire text of the variant forms listed above, only the "actual address".

NS Mail

The NS implementation of mail protocols is obtained by loading the library file NSMAIL.DCOM.

NS addresses are of the form "name:domain:organization", e.g., "Carstairs:PARC:Xerox". If you omit the domain and/or organization, they are defaulted to your own domain and organization. NS mail does not support the many variant forms of address that Grapevine does, because the header is not actually sent as text, but as a structured set of formally named recipients.

Some users of the NS Mail system send messages in a format that Lisp does not understand. The current Lafite implementation leaves such messages in your inbox to be retrieved by other means; all that you see back from GetMail is a header and a note that there was an attachment that Lafite couldn't read.

Changing the Lafite User—Lafite customization

`(LAFITE` *ON/OFF MAILFOLDER* . *OPTIONS*`)`

Used for starting and stopping Lafite with an optional mail folder. If *ON/OFF* is the atom `ON` then Lafite will start processing using *MAILFOLDER*. If *MAILFOLDER* is not supplied then Lafite will use your `ACTIVE.MAIL` mail folder (actually, the value of `DEFAULTMAILFOLDERNAME`). If *MAILFOLDER* is supplied but is the atom `NIL` then no browser will be created but you can send messages and start browsing mail files later using the 'Browse' menu item.

If *ON/OFF* is the atom `OFF` then Lafite will close all mail folders, expunging all messages marked for deletion; close all windows associated with Lafite; and remove the mail watching process from the active process list. This is the same as invoking 'Quit' in the Lafite Status Window.

The third and subsequent arguments are options. The only one currently recognized is the atom `SHRINK`, which instructs Lafite to immediately shrink the browser window brought up on *MAILFOLDER*. In this case, if *MAILFOLDER* is (explicitly) `NIL`, it still defaults to `DEFAULTMAILFOLDERNAME`, since `SHRINK` makes no sense otherwise.

Lafite uses its own host and directory names for mail folders, `LAFITE.PROFILE`, etc., rather than the current connected directory because you may want to keep your mail folders someplace special (e.g., the local disk or your login directory), and the connected directory can change. The global variable `LAFITEDEFAULTHOST&DIR` is provided to tell Lafite where you generally keep your mail. `LAFITEDEFAULTHOST&DIR` should be atomic, in the same form as `LOGINHOST/DIR` (e.g. `{PHYLUM}<CARSTAIRS>`). If `LAFITEDEFAULTHOST&DIR is NIL`, then the value of `LOGINHOST/DIR is used`—i.e., your login host and directory.

If you are running in a Lisp in which some previous user has been using Lafite, you need to take some action to get Lafite to work on your mail files and in your name. When you change your user identity by calling `(LOGIN)` to log in as yourself, Lafite notices that the current user has changed, and attempts to authenticate the new user. However, Lafite still doesn't know how you want your Lafite customized; in particular, Lafite notices the value of `LAFITEDEFAULTHOST&DIR`, and loads your profile of known folders only when Lafite is first started. Thus to change the identity of the Lafite user, you should follow the following procedure:

1) Turn Lafite off, by clicking Quit in the status window, or calling `(LAFITE 'OFF)`.

2) Log in as the new user by calling `(LOGIN)`.

3) Set `LAFITEDEFAULTHOST&DIR` and/or `LOGINHOST/DIR` as appropriate, and any other personal variables that affect Lafite that matter to you. A typical way to do this step is to call `(GREET)` to get your personal initialization loaded.

4) Restart Lafite by calling `(LAFITE 'ON)`.

When things go wrong...

Lafite tries fairly hard not to break, but it is running in a very open environment, where users are free to interrupt arbitrary processes, network servers come and go, etc. As of this writing, the following are some abnormal states of note:

Mail file does not parse. This shows up when you browse a folder, Lafite starts parsing the folder, and encounters an error in the file, typically an incorrect message length. Lafite prints a message to this effect in the browser window and aborts the browse. The information Lafite prints includes the header of the last message parsed, and a byte pointer where the problem was encountered. The byte pointer is such that if you truncated the file to that pointer position, the mail file would be valid.

Solution: scavenge the mail file. The Library package `MAILSCAVENGE` contains a mail file scavenger. The Laurel MailFileScavenger program also works on Lafite files, as does the Hardy scavenger (I think).

The most common way to get a mail file into an inconsistent state is to abort a MoveTo or Update command somewhere in the middle (either manually, or because a remote server crashed). In the case of MoveTo, the problem is usually that the last message in the file is "too short", since it never was completely written. If the first operation you perform on the destination file after the crash is to browse it, Lafite will (usually) detect this situation and let you browse the file anyway, with a warning that the last message is truncated. Updating the file will then correct the length of the last message. Thus, in this case, you will not need a scavenger. If you neglect to browse the destination file before moving additional messages to it, however, you will need to scavenge.

Table of contents inconsistent with mail file. In theory, this should never happen; however, practice shows that it does. Lafite breaks with a message to this effect when it tries to operate on a message that isn't where it thought it was in the file. The appropriate action to take is to close the browser, selecting **Don't Update**, delete the table of contents file (the file with `-LAFITE-TOC` appended to the name), and then browse the file again. If this is not successful, you may need to scavenge the file. If you had made many changes to the browser (deletions, for example) that you would rather not lose, you can try selecting **Write out Changes Only** when you close the browser; this may succeed if the inconsistencies in the table of contents did not intersect with your changes.

Slow file server. If your mail files live on a remote file server that is particularly unresponsive, it may happen that the mail server connection over which new mail is being retrieved times out before the file server acknowledges receipt of the messages. The usual consequence of this is that your inbox is not flushed, so your new mail is in two places: your inbox, awaiting retrieval, and your mail file, to which it was just retrieved. A less common occurrence is that the mail server times out partway through the retrieval process, resulting in a Lisp break. You can ↑ out of the break to return to the state before the GetMail started.

If this is often a problem for you, you may want to adopt the following idiom, which several people have found useful in maintaining the flexibility of remote mail files while utilizing the speed and reliability of the local disk. Keep most of your mail files on the remote server, as usual, but keep your "active" mail file, the one to which you standardly retrieve mail, on your local disk. Retrieve mail to this file, and dispatch from there to your remote files (using MoveTo) some or all of the messages you wish to keep. Mail files on `{DSK}` have very predictable performance during GetMail, which is good for both you and the mail server. Files on `{DSK}` are also less subject to other vagaries of remote servers (e.g., sudden crashes) that sometimes cause problems with mail files. And if you tend to delete much of your incoming mail after reading it once, you will find it much faster to keep your active mail on `{DSK}`, even if your remote server isn't flaky.

You can also choose to keep most or all of your mail files on `{DSK}`, backing them up to a file server periodically. The LispUsers package `COPYFILES` is helpful for doing the backup automatically.

Lafite Profile Variables

Below are the global variables that control Lafite's behavior.

`DEFAULTREGISTRY`

The name of your local registry, used if your login name does not include a registry. This is normally set in the local site-specific `INIT.LISP` file.

`LAFITEDEFAULTHOST&DIR`

The directory on which Lafite looks for `LAFITE.PROFILE` and all mail folders and message forms you access if you don't supply an explicit directory. See "**Changing the Lafite User**'' above.

DEFAULTMAILFOLDERNAME

If no mail folder is supplied to the function LAFITE, i.e., you call (LAFITE 'ON), then the value of this variable is used. Initially, ACTIVE.MAIL.

LAFITEMAIL.EXT

The default extension for names of mail folders. Initially, MAIL.

LAFITETOC.EXT

The string appended to the name of a mail folder to produce the name of its table of contents file. Initially, -LAFITE-TOC.

LAFITEFORM.EXT

The default extension for names of user-defined form files. Initially, LAFITE-FORM.

LAFITEFORMDIRECTORIES

A search path for Lafite forms, initially NIL. When you choose the **Saved Form** command underneath **Send Mail**, the form name that you enter is first searched for on your default directory (LAFITEDEFAULTHOST&DIR), and if not found there, Lafite searches the directories in the list LAFITEFORMDIRECTORIES. LAFITEFORMDIRECTORIES is typically set to a list of one or more public directories on which generally useful forms have been collected.

MAILWATCHWAITTIME

The number of minutes between polling for new mail from your mail servers. Initially set to 5.

LAFITEFLUSHMAILFLG

If NIL, Lafite won't flush your inbox when retrieving new mail, so the mail will still be there when you invoke Get Mail again. Initially, T.

LAFITENEWPAGEFLG

If T, then the Hardcopy command will start each message on a new page. Otherwise it will separate each message by a line of dashes. Intially, T.

LAFITEHARDCOPYBATCHFLG

If you often request hardcopy of single messages, one at a time, you may notice some disadvantages: short messages sometimes get lost in amongst other users long output, and they are wasteful of paper. And if you hardcopy large messages, you may not always care to wait around while the message is formatted for hardcopy. LAFITEHARDCOPYBATCHFLG is provided to allow you to postpone the hardcopying until it can be done all at once.

When this flag is true, Lafite "batches" your hardcopy requests, and doesn't actually print them until you do an Update, at which point it sends them all to the printer in one batch. When you have hardcopy pending, the Hardcopy button is speckled to remind you of this fact. The Update button has an additional choice **Do Hardcopy Only** in case you want to get your batched hardcopy printed without doing an actual Update.

The behavior of LAFITENEWPAGEFLG when batching hardcopy is that it applies only to the messages selected at each Hardcopy invocation; each new set of messages starts on a new page, independent of the setting of LAFITENEWPAGEFLG.

LAFITEHARDCOPYBATCHFLG is initially NIL.

LAFITEHARDCOPY.MIN.TOC

If non-NIL, is a positive number. Whenever Lafite is instructed to produce hardcopy for more than LAFITEHARDCOPY.MIN.TOC messages, it also produces a table of contents as a cover page for the hardcopy. Currently, this flag is only noticed if LAFITEHARDCOPYBATCHFLG is NIL. Initially NIL.

LAFITEDISPLAYAFTERDELETEFLG

If T, Lafite will display the next message if you delete the one that is in the message display window and the next message is undeleted and has not been examined (i.e., it is marked with a "?"). If ALWAYS, then it will display the next undeleted message even if it has already been seen. Initally, T. T is roughly Laurel semantics, ALWAYS is Hardy semantics.

LAFITEMOVETOCONFIRMFLG

Controls whether Lafite requires confirmation of the Move To command. If ALWAYS, all moves require confirmation; if LEFT, then only left-button moves (selecting the destination from a menu) require confirmation; if MIDDLE, then only middle-button moves (using the "default Move To" folder) require confirmation; if NIL, then no moves require confirmation. Initially ALWAYS.

LAFITEBROWSERREGION, LAFITEDISPLAYREGION, LAFITEEDITORREGION

These are REGIONs which are used to describe where the primary (i.e. first) window of each type is to be placed on the screen. Initally, they are set to something "reasonable" for the standard initial display configuration. If you set them to NIL then you will be asked to specify a region (via GETREGION) the first time any such window is created.

LAFITEDISPLAYFONT, LAFITEEDITORFONT, LAFITEHARDCOPYFONT

These are the fonts used for displaying messages, composing messages, and making hardcopy. You may change them individually. They should be FONTDESCRIPTOR's as returned by FONTCREATE. Initially, they are all TimesRoman12.

LAFITEBROWSERFONT

The font used for displaying the table of contents in the browser window. Initially, Gacha10.

LAFITEMENUFONT

The font used for the items in all Lafite menus. Initially Helvetica10 Bold.

LAFITETITLEFONT

The font used for the title bar ("Lafite") in the Lafite status window. Initially Helvetica12 Bold.

LAFITESTATUSWINDOWPOSITION

Specifies where the status window appears when Lafite is invoked. It is a POSITION or NIL (in which case you will be asked to specify a position when Lafite starts).

LAFITEOUTBOXSIZE

Specifies the number of delivered messages that should be retained in your outbox. As you send more messages, older ones fall off the end. Increasing this number gives you a longer

”history” from which you can select and re-edit old messages, but this desire should be balanced with the knowledge that you are tying down the resources used by each of those messages. Setting LAFITEOUTBOXSIZE to zero or NIL disables the outbox feature: after delivery, messages completely vanish. LAFITEOUTBOXSIZE is initially 2.

LAFITENEWMAILTUNE, LAFITEGETMAILTUNE

(Dandelion only) These are lists of the form acceptable to the function PLAYTUNE, or NIL, in which case they are ignored. LAFITENEWMAILTUNE is played when Lafite discovers you have new mail waiting; LAFITEGETMAILTUNE is played when a Get Mail command completes.

LAFITEENDOFMESSAGESTR, LAFITEENDOFMESSAGEFONT

LAFITEENDOFMESSAGESTR is a string containing the text of the ”End of Message” token displayed at the end of a message; LAFITEENDOFMESSAGEFONT is the font in which it is displayed. If LAFITEENDOFMESSAGESTR is NIL, then no ”End of Message” token will appear.

LAFITEIFFROMMETHENSEENFLG

If true, then messages sent from you are considered ”Seen” (and hence do not have the mark ’?’), even though you have not yet displayed them. Initially T.

LAFITEBUFFERSIZE

The number of 512-character buffers used by the stream managing the file behind an open browser window. If you regularly receive very long messages, you might want to increase this to improve performance of Display followed by HardCopy or Move. Initially 20, which handles up to about 10,000-character messages.

LAFITEMODEDEFAULT

Determines the mode Lafite comes up in when more than one set of protocol implementations has been loaded. Initially NIL, which means that if there is a choice, Lafite simply waits for the user to choose a mode.

LAFITESHOWMODEFLG

Determines whether Lafite displays the current mode in the status window. If ALWAYS, it always does; if NIL, it never does; if T, it does only if there is any ambiguity (more than one set of protocol implementations has been loaded). Initially, T.

Adding New Message Forms to Lafite

The normal way to add new message forms to Lafite is to edit an existing form (or build one from scratch) and save it away using the ’Save Form’ menu item. You can also provide message forms that compute the text on the fly, as, for example, the ’Lisp Support’ selection does. To add your own items to the ’Message Forms’ menu, add a standard three-element menu item to the variable LAFITESPECIALFORMS and then set the variable LAFITEFORMSMENU to NIL (this is where the menu is cached). The three-element menu item should yield a LITATOM as its ”value”, that atom being interpreted as follows:

1. If the atom has a function definition, the function is called (with no arguments) and the returned value (a string or a TEdit TextStream) is used;

2. If the atom has a value, its value (a string or a TEdit TextStream) is used;

3. otherwise, a copy of the file by that name is used.

For example, if TEdit wanted to add a message form that contained the date the TEdit was made (similiar to Lafite's bug report form) it could add

```
("TEdit Support" (QUOTE MAKETEDITFORM)
           "Make a form to report a problem with TEdit")
```

to `LAFITESPECIALFORMS`; `MAKETEDITFORM` could be defined to be

```
[LAMBDA NIL
  (PROG (OUTSTREAM)
        (SETQ OUTSTREAM (OPENTEXTSTREAM ""))
        (printout OUTSTREAM "Subject: TEdit: >>Subject<<" T)
        (printout OUTSTREAM "To: " TEDITSUPPORT T)
        (printout OUTSTREAM "cc: " (USERNAME) T)
        (printout OUTSTREAM "TEdit-System-Date: " TEDITSYSTEMDATE T T)
        (printout OUTSTREAM ">>Message<<" T)
        (RETURN OUTSTREAM]
```

where `TEDITSUPPORT` and `TEDITSYSTEMDATE` are variables set by TEdit. Lafite supplies one function to make this kind of message form easier to construct:

`(MAKEXXXSUPPORTFORM SYSTEMNAME ADDRESS SYSTEMDATE)`

Creates a message form (a TEdit stream) to be mailed to the maintainers of `SYSTEMNAME`. `SYSTEMNAME` is the name of the subsystem (a string); `SYSTEMDATE`, if non-`NIL`, is a date (string) of importance to include in the message; and `ADDRESS` is the mail system address of the intended recipient(s).

For example, if `MAKETEDITFORM` were defined as

```
[LAMBDA NIL
  (MAKEXXXSUPPORTFORM "TEdit" "TEditSupport" TEDITSYSTEMDATE)]
```

Then selecting "TEdit Support" in the Message Forms menu would produce a form such as the following:

```
Subject: TEdit: >>Subject<<
To: TEditSupport
cc: vanMelle.pa
TEdit-System-Date:  3-Feb-84 12:23:49
Lisp-System-Date:  3-Feb-84 18:13:22
Machine-Type: Dorado

>>Message<<
```

There is a new and substantially revised version of Lafite on [Phylum]<Lisp>Library>Lafite.dcom and in [Phylum]<Lisp>Current>Full.SYSOUT. If you want to load the new Lafite on top of an old Lafite, please Quit the old Lafite first.

The changes in this version are mostly in the internal implementation, rather than the user interface, but there are a few user-visible changes. The major such change is that Lafite no longer attempts to keep your mail browser and the mail file itself consistent with each other at all times. In particular, changes (deletions, marks changed) only get written out to the file when you do an Update, and "MoveTo"s are only guaranteed to have finished then, too. To make this more feasible, there are now two flavors of update: "Update by writing out changes only" and "Expunge deleted messages". The latter is like the old Update, the former can be thought of like a SAVEVM: it does not compact out deleted messages, but does write out changed marks and an updated contents file, assuring that if you booted your machine after the Update that you could browse the mail file again and get back to your previous state.

When you close or shrink a browser that has changed in any way needing update, you get a menu of choices--the two flavors of update, or no update at all.

The value of the variable LAFITESTATUSWINDOWPOSITION is now the lower left corner of the entire window, as you would expect, rather than the lower left corner of the "You have new mail" part of it.

The variable LAFITEUSEHIGHESTVERSIONFLG has gone away. Lafite always reads the highest extent version of a file, and Update goes to that version. That version is, of course, ;1 if you are a using Laurel on the same disk.

There is a completely new table of contents format, which is more compact and allows a more efficient update, one that is only proportional to the number of changed messages, rather than to the total number of messages. If you have an old table of contents for a file you browse with the new Lafite, it will be discarded.

Assorted bugs have been fixed, including some relating to resuming over Logout or SaveVm, or clicking GetMail too soon. I would be surprised if I'd caught all such bugs, though. If you have some favorite old bug (as opposed to suggestion) that hasn't been fixed, you might want to retransmit a bug report for it.

Many internal data structures have been revised; some algorithms that took time proportional to the number of messages in a file are now constant-time algorithms; there is a completely new message header parser. If you rely on any undocumented features of Lafite, let me know, as they may well have changed.

If you are a Laurel 6.1 user, you may find that Laurel has transformed some spaces in your messages from character 40Q to 240Q as some sort of line-breaking hack. There is a little function COPY7BITFILE that can be used to copy such a file removing the high-order bits. Args are SRCFIL and DSTFIL, with DSTFIL defaulting to a new version of SRCFIL.

There will be updated documentation some day soon.

Complaints and suggestions to LafiteSupport as usual.

Changes to Lafite in the version of January, 1990.

New NS Mail implementation. There is a new module called NEWNSMAIL, which implements the new mail protocols supported in Services 11.2. This module must be loaded in addition to (and after) NSMAIL. From the user's point of view the two implementations are similar, except that with the new protocols you can retrieve parts of messages not available with the old protocols (such messages when retrieved with the old protocols contain a notation "NOTE: Additional parts of this message are available only via the new Mailing Protocols.", and the Lafite status window subsequently over-reports the amount of new mail you have by the number of such messages).

Whether you use the new protocols is controlled by the variable *USE-NEW-NSMAIL*: if true use the new protocols, if NIL use the old. After changing this variable, you must Recache (under middle-button Quit) or restart Lafite in order for it to notice. The variable is initially T.

If any of your mailbox servers does not support the new protocols, Lafite sets *USE-NEW-NSMAIL* to NIL and starts over. All the servers at PARC support the new protocols.

Changes to Lafite since October, 1988.

There is an additional command on the message composition window: **Change Mode**. This lets you change the mode of a message from GV to NS or vice versa. This can be useful if you clicked **SendMail** while in the wrong mode, or you want to Answer a message but do so from the other side of the fence. This command does not make any attempt to fix addresses already in the header to their other form; you have to do this yourself. The one exception is that it will change "cc: *Your old-mode name*" to "cc: *Your new-mode name*".

There is a new command on the middle-button titlebar menu of the browser: **Cancel Pending Hardcopy**. Of course, this is only interesting when LAFITEHARDCOPYBATCHFLG is T.

You can now send to private dl’s in NS mode as well as GV mode. The syntax is the same: "To: dlname:;". In NS, this parses as the NS name "dlname:;:Xerox" (and will look like that to the recipients), but Lafite recognizes this kind of address and instead looks for a file named dlname.ext on your Lafite directory.

Changes to Lafite during February, 1989.

There is a new kind of filter for NS recipients of GV messages. If you include the symbol GV in the list LAFITE.DONT.DISPLAY.HEADERS, then messages sent from GV to NS will have their ugly "GVGVGVGV..." section hidden (this section includes any headers that didn't translate into NS land, such as comments in addresses ("Fred Carstairs <fc@foo.bar.com>") and all those awful "Received:" lines in arpa mail. As usual, the Unhide command on the window will reveal the truth. You can also add GV to LAFITE.DONT.FORWARD.HEADERS and LAFITE.DONT.HARDCOPY.HEADERS.

Lafite canonicalizes your LAFITEDEFAULTHOST&DIR. This should reduce confusion among users living on NS servers, who might have set the variable to, say, "{EG:}<Carstairs>Mail>", only to have Lafite confused when file names came back in the form "{EG:PARC:Xerox}...".

Lafite handles the form files menu better.

The "Reply-To" and "Send Formatted?" menus no longer step on each other if you deliver two messages requiring them at the same time.

MAKEXXXSUPPORTFORM, when given an address in a-list form, chooses the first address in the list that is for a supported mode. Thus, we can arrange to have LISPSUPPORT always go to an NS address (assuming NS mail loaded). Special forms that call MAKEXXXSUPPORTFORM can also encourage a particular mode this way without having to switch the mode themselves.

NS mail no longer holds a session open on the mail server in certain odd cases.

NS mail header fields are displayed in the order specified by the variable NSMAIL.HEADER.ORDER, independent of the order in which the mail system may have delivered them (the new Services 11 servers deliver some headers in an odd order).

Changes to Lafite in the version of May, 1989.

New files. There are 3 new files: LAFITEFOLDERS, LAFITESORT, MAILSCAVENGE. In addition, Lafite requires the file DATEPATCH for improved date parsing. These are all loaded automatically by LAFITE.

Folder hierarchy. [This feature derives from a package written by Mike Dixon.] You can now organize your folders into a hierarchy, so that the folder menus presented by **Browse** and **MoveTo** are easier to use. Folders are organized into *groups*. A folder can be in any number of groups. Groups can have subgroups. The top level of the folder menu is composed of all the groups that are not a subgroup of some other group, plus all folders that are not in any group. The **Edit Folder Hierarchy** command on middle-button Browse lets you define new groups and change existing ones.

Note: the folder hierarchy is saved on your Lafite.info file, along with folder names and form names. Older versions of Lafite do not know about this structure, but those of the last year will at least preserve it. However, if you delete folders and then come back to a new version of Lafite, the deleted folders may still appear in the hierarchy; you must manually delete them either by editing the variable LAFITE.FOLDER.STRUCTURE or by editing the file Lafite.info.

Sorting. There is a new command on the middle-button browser titlebar menu: **Sort by Date**. This command sorts the messages in the browser in order of their "Date:" fields. Messages lacking a Date field or having an unparseable Date field are sorted so as to remain next to the preceding message. There is a subcommand **Sort Selected Range**, which only sorts between the first and last selected messages. In either case, the messages are sorted in the browser only, until you next **Update**, at which time the messages are written to the file in the sorted order. When messages have been rearranged, the **Update** option **Write Out Changes Only** is not available, as there is no file format for out of order messages; you must **Expunge**, even if there are no deleted messages.

Sorting New Mail. You can have Lafite automatically sort new mail by setting the variable LAFITE.SORT.NEW.MAIL. If the value is T, Lafite always sorts new mail; if :MULTIPLE, Lafite only sorts when the mail was retrieved from more than one server. Note that even when using a single mail server, mail can easily be out of order, due to the varying speeds at which messages pass through gateways, etc. Sorting assures that the messages will appear in the order in which they were posted. Sorting new mail seems to be fast enough (at least on my Dorado) that it is probably always worth doing.

Dates. As you might guess from the above, Lafite now actually parses the Date field of messages. Aside from enabling Sort, this allows Lafite to display the date in the browser in a canonical form, rather than one that varies with the whim of the sender's software. Lisp's date parser (IDATE) was substantially beefed up to handle a wider variety of dates. It now claims to handle all dates legal in RFC822 syntax (except the silly single-digit military time zones), plus a fair variety of other formats found in mailers of recent years. The parser changes are in a module DATEPATCH loadable separately from Lafite.

Lafite considers dates older than 1970 spurious, so as to avoid being fooled by messages from machines that neglected to set the time.

With a new representation of dates, Lafite also has a new table of contents format. If you browse a folder with an old-style toc, Lafite will print "(older format)" when it reads it. Next time you do an Update, the toc will be written in the new format. Lafite does not automatically parse dates on old messages, so until you issue a Sort command, old message dates will still be displayed as whatever string the old toc saved, not the new canonical form. In addition, when you *do* issue the Sort command, it must first go back and retrieve afresh all the date fields and parse them. Old versions of Lafite cannot read the toc files saved by this new version of Lafite, so if you try to browse a new folder with an old Lafite it will be forced to parse the file from scratch.

Scavenger. The mail scavenger has been completely rewritten. It now handles most simple cases of bad message lengths by altering the file in place, rather than laboriously copying the entire file to a scratch location. The diagnostic output is, I hope, more informative. The scavenger is also integrated with the browser, so that if you browse a malformed folder and the browser reports "unable to parse", you are immediately offered the option of scavenging.

Expunge. Lafite now gets less confused if you abort it in the middle of an Expunge. Either you killed it before it got to the point of irrevocably altering the file, in which case it is as if you had never started the Expunge, or it will report that the file is inconsistent and must be rebrowsed.

Long messages. The Lafite message file format has been very slightly extended to permit you to receive messages of up to 99,999,999 bytes (about 100MB!) in length. This means it no longer is forced to split apart messages longer than 99,999 bytes, which did very bad things to large Viewpoint attachments and sometimes caused loss of mail server connection. It also means that Lafite files are no longer exactly compatible with Laurel or Hardy format, in case anyone cares. As mitigation, however, messages that fit in the old format (99,999 bytes or less) are converted back to the old format whenever they are moved (either by **MoveTo** or **Expunge**), so you can arrange for a Hardy-compatible file by moving all messages into another folder (assuming your messages are all short enough).

NS Mail. Lafite accepts messages sent in serialized format 3, which it turns out is identical to format 2.

Lafite now sends plain text messages as "text attachments" rather than "mail notes", unless *NSMAIL-SEND-MAIL-NOTES* is true. This avoids a nasty bug in Services 11.3 mail servers.

An incomplete and not very well tested feature: you can send attachments. The attachment must be a file on an NS file server. To send an attachment, place a line "Attached-File: *XNS Filename*" in the header of your message. To send the file as a reference (this sends only a pointer to the file, not the file's content, so the recipient must have access to your server), use the line "Attached-Reference: *XNS Filename*". You can have only one such line in the header, and the body of the message must be small enough to send as a mail note (less than 8000 characters). Probably the only interesting kind of attachment to send currently is an Interpress master, which both Lafite and Viewpoint recipients will be able to print. To "forward" an attachment you received in NS mail, you can use **Put to File** on the attachment, then compose a message with "Attached-Reference" or "Attached-File" referring to the resulting file. Incompleteness: Most of the attributes of the file are lost when you send the file itself as an attachment, rather than a reference. You cannot send directories as attachments.

Changes to Lafite in the version of August, 1989.

The major changes in this version all affect NS mail:

You can now send NS mail even if you don't have an NS mailbox, as long as you have an NS identity. You must be sure you have forwarding from this NS identity, however, since otherwise there is no place for the mail system to send failure replies.

The system automatically detects when your mailbox has been moved from one server to another and adjusts accordingly, instead of reporting the server's incomprehensible error message "ACCESS.ERROR NoSuchRecipients".

The names in the To and Cc fields of NS messages are now abbreviated where possible. For example, if the sender is Burwell:PARC:Xerox, then recipient Carstairs:PARC:Xerox is displayed as "Carstairs", and ClockWatchers:All Areas:Xerox as "ClockWatchers:All Areas".

The "View as Text" command on NS attachments is slightly smarter about filtering out binary garbage, and recognizing the Viewpoint end of line character. The window that "View as Text" uses is automatically closed when you display the next message or shrink/close the folder (if you want to retain it, you have to use the middle button to display the message containing the attachment).

Changes to Lafite since July 19, 1988.

The browser middle-button title-bar menu has three new commands (in addition to the Find commands): Go to # lets you jump to a message by number; Describe Folder prints some informative information about the folder; and Enable Move To Menu attaches a menu to your browser window to let you move messages quickly to other folders. Setting LAFITE.AUTO.MOVE.MENU true lets you get this accelerated menu automatically.

Middle-button on an "active" browser’s icon now only offers the "Get Mail" menu while the button is held down; if you don’t select "Get Mail", it simply expands the window. This means that middle-button on that icon is more like every other icon.

The Browse subcommand "Forget Folders" lets you select lots of folders at once, somewhat more gracefully than the old way.

There is a new Browse subcommand Rename Folder.

Host abbreviations: if you have mail files on multiple directories, there is now an abbreviation facility. See documentation of LAFITE.HOST.ABBREVS.

The bug wherein Lafite forgot LAFITEDEFAULTHOST&DIR (and other "Personal" variables) over logout is (I hope) fixed.

Bug reports include the sysout name in addition to its date.

Changes to Lafite in the version of November, 1989.

New Looks Subcommands. On the Middle-button menu in a Lafite message display window, the Looks command has, in addition to **Default** and **Fixed Width**, three new subcommands:

Lowercase	Converts message body to lowercase. This is useful for messages from those bozos who haven't discovered their Shift Lock yet, or are typing on Model 33 Teletypes.
Spread Paragraphs	Inserts 10-point leading between paragraphs. This is useful for messages that come from Tioga, which doesn't put a blank line between paragraphs in the plain text body.
VP Line Breaks	This turns the character #o35, which at least some mail senders in the XNS world seem to use as an end-of-line character, into carriage return. Maybe some Viewpoint expert could fill me in.

Saved Forms. The **Save Form** button on a message sending window is now sanctioned to save (checkpoint) messages in progress, as most of you seem to want to do much more than create private mail forms. In light of this policy shift, the button has been renamed simply **Save**, and the message window stays open afterwards (close it as you wish). When a previously saved message is ultimately delivered, you are asked whether you want to keep the saved form or delete it. There is also a menu item **Delete Message Form** underneath middle-button **Browse** for deleting a saved message explicitly. You can think of **Save** as a fancy version of TEdit's Put, which stores the message in your mail directory and remembers it (in the Saved Form menu) for a later Get.

Reply To. There is an additional button labeled **Reply To** on message sending windows. It inserts a "Reply-to: *yourname*" field in the message being composed. The name is selected for pending delete, so if you want replies to go to someone else, you can just start typing the new name.

Rename Folder now has a better prompt. It also takes no action if you just pressed carriage return, thus returning the old name.

Bad toc. Fixed error message when you read a malformed toc (previously it broke). Several people have run into this problem by copying toc files between Sun and NS file servers on Sun Medley, which erroneously called the toc files type TEXT.

Scavenger. The programmatic entry to the mail scavenger has been renamed LAFITE.SCAVENGE, to avoid confusing people who knew the old argument list. Ordinarily, you should never have to call the scavenger by hand, as the **Browse** command will invoke it for you when needed.

I assume you are willing to deal with a completely unpolished interface, so I'll tell you what the current truth is about the Lafite on <LispCore>Library> and in the latest <LispCore>Next>Full.sysout. What follows might be viewed as the start of an implementor's manual. It is not, of course, anything I'd be willing to publish as a programmer's manual, but perhaps your experience will help us define what one might wish the interface really to be. I'm quite willing to add small things as you see the need, though there is little hope for any major projects in the near term. So let me know how this matches your needs.

My other reluctance, of course, is that internals, as these all are, are subject to change without notice, so I'd appreciate feedback on what parts you would like to see solidified into a supported interface.

There are almost no direct functional interfaces to what you want, but with some record definitions I think you can get most of it. You thus need to LOADCOMP(LAFITE). In general, you should probably avoid replacing record fields, as there are usually interactions; if you need to replace a field for which there is no programmatic interface, let me know. Fetching fields should be fine. The important records are two datatypes, MAILFOLDER and LAFITEMSG.

MAILFOLDER holds assorted facts and state about a single mail folder. The window property MAILFOLDER hangs off each Lafite browser window. Here are the most interesting fields:

FULLFOLDERNAME, VERSIONLESSFOLDERNAME, SHORTFOLDERNAME -- various forms of the folder's name.

FOLDERSTREAM -- a stream on the file behind the folder, if it's open.

MESSAGEDESCRIPTORS -- an array of LAFITEMSG objects, corresponding to messages 1 thru #OFMESSAGES. Thus (ELT (fetch (MAILFOLDER MESSAGEDESCRIPTORS) of folder) n) returns the nth LAFITEMSG in folder. This field is only valid while a browser is open on the folder.

FOLDERLOCK -- a monitor lock that should be acquired if you want to perform any operations on the folder.

FIRSTSELECTEDMESSAGE, LASTSELECTEDMESSAGE -- number of the first and last messages currently selected (equal if only one selected).

BROWSERWINDOW, BROWSERMENU, BROWSERMENUWINDOW, BROWSERPROMPTWINDOW -- pointers to parts of the browser window for this folder. NIL if no browser open on folder.

(\LAFITE.GETMAILFOLDER folderName) -- this function returns a MAILFOLDER object for folderName, which should be either a full file name or a full name sans version number. If the result has a non-NIL BROWSERWINDOW field, there is a browser currently open on the folder, else not.

(\LAFITE.OPEN.FOLDER folder access recog) -- returns a stream on the file behind folder. access and recog are as with OPENSTREAM. This is the standard way to get the stream (rather than the FOLDERSTREAM field), unless you're sure the file's open for the right access.

Each message in a browser is described by a LAFITEMSG record. Its most interesting fields:

PARSED? -- true if message has been parsed. Set by LAFITE.PARSE.MESSAGE.FOR.TOC, which also distributes the standard parse into the fields FROM, SUBJECT, DATE.

DELETED? -- true if message is marked deleted.

SEEN? -- true if message is marked seen (NIL => "?" in browser).

MARKCHAR -- character code of message's mark.

MARKSCHANGED? -- true if MARKCHAR has changed since last update.

SELECTED? -- true if message is currently selected.

MSGFROMMEP -- true if message is from the logged in user.

# -- ordinal position of message in folder, from 1.

BEGIN -- byte position in file of start of message, which is always the internal header that starts "*start*".

MESSAGELENGTH -- length of message, including internal header.

STAMPLENGTH -- number of bytes between the BEGIN position and actual start of message.

START -- access field: byte position of the start of the actual message, viz., BEGIN+STAMPLENGTH.

END -- access field: byte position of the end of the message, viz., BEGIN+MESSAGELENGTH.

DATE -- date of message, actually just the 6 character string that appears in toc (so you can't give it to IDATE).

FROM, SUBJECT, TO -- From, Subject, To fields of the message. The TO field is usually only parsed out if MSGFROMMEP has been observed to be true.

Now, how to use all these:

The value of the variable LAFITE.AFTER.GETMAIL.FN, if non-NIL, is a function to call immediately after Lafite retrieves mail from a server. It is called with two arguments, the MAILFOLDER, and a list of LAFITEMSG objects describing the new messages (which have already been appended to MAILFOLDER:MESSAGEDESCRIPTORS, securely stored in the file, but not yet displayed in the browser window). The messages have not yet been parsed, I believe. The function is called while in control of MAILFOLDER's monitorlock.

Parsing messages. The function LAFITE.PARSE.MESSAGE.FOR.TOC sets all the fields in a LAFITEMSG that the toc needs, but you probably want more fields than that, so will need to do your own parsing. Lafite has a simple-minded but reasonably efficient parser that looks for header fields in a message (Field name followed by colon) and does something with them. It requires that you build a "parse table" describing what you want it to do. This is a list of lists, one for each field you want to parse. Each sublist is in the form ("FieldString" ResultFn . ResultArgs). Then "compile" the parse table by calling (LAFITE.MAKE.PARSE.TABLE table). The resulting compiled table can then be passed to this function:

(LAFITE.PARSE.HEADER stream parseTable start end onceOnly) -- Stream is the stream where the message lives -- the folder's stream for message in a file, or a tedit stream for a message you're sending. parseTable is the compiled parse table. Start and End are byte pointers in stream (normally the START and END fields of a LAFITEMSG). Parsing automatically stops at the end of the header. onceOnly is true if you want parsing to end as soon as any one field has been successfully parsed.

The parse proceeds as follows: the parser starts at byte position start and scans each line in the header, seeing if the field name matches (case-insensitively) a field in the parse table. If a match is found (the characters match up thru the colon), then the parser skips over any white space, then calls the ResultFn indicated in the parse table with two arguments, the stream (now positioned at the first non-blank character after the colon) and the ResultArgs from the table (this lets you multiplex one parsing function for several purposes). The function's job is to parse the rest of the field, leaving the stream positioned after the terminating <eol>, and to set freely the variable PARSERESULT as it wishes. PARSERESULT is what LAFITE.PARSE.HEADER ultimately returns.

You may not need to follow that too closely. Lafite's standard use of the parser is to make PARSERESULT an alist; the following functions are interesting as resultFn's, or to help them. You can look at the variable LA.FULLPARSEFIELDS and LA.TOCFIELDS for examples. The former is used for a "full parse", such as the Answer command needs; the latter parse three fields out for the toc.

(LAFITE.READ.TO.EOL stream) -- Reads the rest of the field, returning one long string. You should probably use this as the basis for all result fns, as it takes care of continuation lines and makes sure the stream is left properly at the end of the field.

(LAFITE.READ.LINE.FOR.TOC stream Args) -- Reads the field as one big string and stashes it on PARSERESULT indexed by (CAR Args). If all of your resultFn's are this one, then your parse result ends up something like ((field1 "contents1")(field2 "contents2")...). The string is limited in length to 255 characters (a toc restriction); if longer, it is truncated.

(LAFITE.READ.NAME.FIELD stream Args) -- This is almost the same, but no string length restriction, and it recognizes that some fields may have multiple occurrences (bad form, but allowable). If a field occurs twice, its alist entry has more than one string in its cdr: ((field1 "contents1a" "contents1b") ...).

(\GV.PARSERECIPIENTS field registry internalFlg editWindow) -- This function takes raw strings from the output of LAFITE.PARSE.HEADER and produces a list of individual recipient names. Field is a string or list of strings, such as LAFITE.READ.NAME.FIELD might produce during the header parse. registry is the grapevine registry default for names without registries (defaults to DEFAULTREGISTRY). If internalFlg is true, the names returned are in internal form (name . reg); if NIL, the names are strings of the form "name.reg". editWindow is for error messages (true when sending mail).

(LA.SETDIFFERENCE x y) -- like LDIFFERENCE, but treats x and y specifically as sets of strings, case-insensitively.

(LA.REMOVEDUPLICATES x) -- removes duplicates from x, treating x as a set of strings, case-insensitively.

Message manipulation. Most of these update the browser window appropriately.

(SELECTMESSAGE msgDescriptor mailFolder) -- "selects" msgDescriptor, a LAFITEMSG object, in mailFolder. Does not deselect anything.

(UNSELECTALLMESSAGES mailFolder) -- unselects all messages.

(MARKMESSAGE msgDescriptor mailFolder mark) -- changes msgDescriptor’s mark character to mark (a charcode).

(SEENMESSAGE msgDescriptor mailFolder) -- mark msgDescriptor "seen".

(DELETEMESSAGE msgDescriptor mailFolder) -- mark msgDescriptor "deleted".

(UNDELETEMESSAGE msgDescriptor mailFolder) -- undelete msgDescriptor.

Most of the browser commands consist of a function \LAFITE.nameOfCommand, some of which spawn processes named \LAFITE.nameOfCommand.PROC. Here’s the one for MoveTo:

(\LAFITE.MOVETO.PROC window mailFolder destinationFolder item menu shortOutput oldFileP) -- moves all selected messages from mailFolder to destinationFolder. Greys out item of menu while it’s working (the BROWSERMENU field of mailFolder gets you the menu). When finished, if exactly one message was moved, and it was the one currently displayed, does the "display after delete" action. shortOutput is the folder’s name (in short form, but it really doesn’t matter). oldFileP is true if the caller believes, but has not yet verified, that destinationFolder describes an existing folder (this is true if the folder was on the folder menu). window is mailFolder’s BROWSERWINDOW.

As I said, "unpolished". You probably really want the not yet defined subfunction of that: (\LAFITE.MOVE.MESSAGES mailFolder msgDescriptors destinationFolder).

LAFITE UPDATE

This document describes changes to Lafite since the October, 1986 release.

Lafite Modes

Lafite is now willing to operate in more than one mode "simultaneously". The mail watch background task checks all of your mailboxes in all the modes that are active, and **GetMail** retrieves mail from all mailboxes**.** The variable `LAFITE.USE.ALL.MODES` controls whether Lafite runs in more than one mode at once. The interesting values are:

`NIL`	This is the old way, where you must change modes manually. All commands operate only in the currently active mode.
`:POLL`	The mail watch task checks all modes, but **GetMail** only retrieves in the currently active mode.
`T`	Mail watch checks and **GetMail** retrieves in all modes.

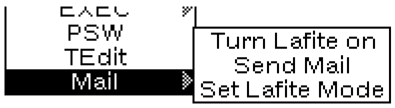
The initial value is `T`. The flag has no effect on the **SendMail** command, which continues to bring up a message composition window in the current mode. It also does not affect the **Answer** command, which answers in the mode that the message was retrieved in. The retrieval mode is remembered in the table of contents file; in the absence of this knowledge, e.g., if the toc is deleted or the message was retrieved with an older Lafite, the **Answer** command uses simple rules that nearly always unambiguously distinguish GV from NS messages. Be aware that the value `:POLL` can be confusing to such hacks as automatic mail retrievers, since the status window can report new mail, yet **GetMail** won't retrieve it.

Ordinarily, Lafite only attempts to authenticate you in each mode when it starts up, and after logout. If, for example, the NS authentication service is down when you start up Lafite, or your NS password is incorrect, it does not repeatedly try to authenticate you, but merely reports the condition once and behaves as if the failing mode does not exist. If conditions later change, you can force another attempt by using the mode switching commands to explicitly switch to the mode. If your password is incorrect, you can change it using the NS Login subcommand of Quit (see below).

GetMail only attempts to retrieve mail from the servers that reported new mail in the most recent poll. This means that if one of your mail servers is not responding, you no longer have to wait during **GetMail** for it to time out. If you click **GetMail** at a time when the status window reports "No New Mail", then **GetMail** will attempt to contact all of your mailbox servers, just as it did in previous releases.

Turning Lafite on and off

You can now turn Lafite on or switch modes from the background menu.



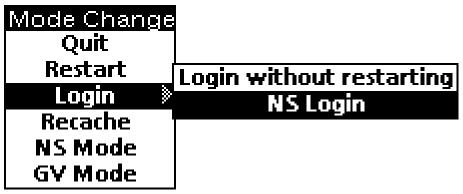
Mail This brings up a standard message sending window in the current mode, whether Lafite is on or not. If Lafite is not on, it will attempt to authenticate you in the current mode.

Send Mail This is the same as the main **Mail** command, except that you get to choose which kind of message form, just as middle-button on the Lafite status window's **Send Mail** command allows.

Turn Lafite On This is the same as calling `(LAFITE :ON)`. It has no effect if Lafite is already on.

Set Lafite Mode This command allows you to change Lafite's mode. It offers a menu of the mode choices found also as part of the middle-button **Quit** command on the Lafite status window.

The middle-button **Quit** command on the Lafite status window has two new subcommands, **Recache** and **Login**. Here, for completeness, is a description of all the commands:



Quit This is the same as the main command, which is the same as calling `(LAFITE :OFF)`. It closes all folders, asking about updating them if any need it, then turns off Lafite.

Restart This turns Lafite off, then immediately back on; same as `(LAFITE :RESTART)`.

Login Changes the logged-in user for Lafite. This command first turns Lafite off, so that any updating required by closing out the previous user's folders happens under the previous user's login. It then prompts for user name and password, exactly as if you had called `(LOGIN)` yourself, and then for a new `LAFITEDEFAULTHOST&DIR`. Finally, it turns Lafite back on.

Login without restarting Performs a login without changing the state of Lafite. This is useful if you had previously logged in with an incorrect password, or if you have multiple user identities all sharing the same set of folders.

NS Login Prompts for a new login just for NS mail. You may need to use this if your global user name does not correspond to your NS user name or an alias of it in the default domain, or if you have different passwords for Grapevine and NS. There is actually one of these commands for each mode that understands an independent login, but NS is the only current instantiation.

Recache In the case of several of the public interface variables, Lafite computes other structures based on them, and does not automatically recompute them when you change the public variable. The recache command forces Lafite to recompute these "caches" of the variables' values. A recache also occurs whenever Lafite is (re)started. The command is implemented by the function `LAFITE.COMPUTE.CACHED.VARS`, which you can advise if you have other caches that Lafite doesn't know about. The variables currently affected by this command (i.e., changes in whose values are not otherwise noticed) are:

```
LAFITE.DONT.DISPLAY.HEADERS
LAFITE.DONT.FORWARD.HEADERS
LAFITE.DONT.HARDCOPY.HEADERS
LAFITE.EXTRA.DISPLAY.COMMANDS
LAFITE.LOOKS.SUBCOMMANDS
LAFITE.GV.FROM.FIELD.
```

NS Mode
GV Mode Switch to the specified mode. There is one of these for each mode.

Screen Appearance

There are several new variables and one function for controlling the appearance of your Lafite screen.

`LAFITE.BROWSER.LAYOUTS` [Variable]

A list of browser "layouts" specifying where Lafite should place browsers, their display windows and their icons. Each layout is of the form (*BrowserRegion IconPosition DisplayRegion*). *IconPosition* and *DisplayRegion* are optional. The region variables are standard window system regions of the form (*left bottom width height*); *IconPosition* is a position of the form (*x* . *y*). When the **Browse** command creates a new browser window, it chooses the first element of `LAFITE.BROWSER.LAYOUTS` that is not already in use. If all are in use, it prompts for a region.

`LAFITE.DISPLAY.SIZE` [Variable]

Specifies the default size of display windows, other than those already specified in a browser layout. If the value of this variable is a size (*width* . *height*), then Lafite prompts for a region with a box this size. If the value is `NIL`, Lafite prompts for a general region, requiring you to drag from one corner to the opposite. The initial value is `(500 . 300)`.

`LAFITE.EDITOR.LAYOUTS` [Variable]

A list of editor "layouts" specifying where Lafite should place message composition editors and their icons. As message editors have only a single window, each layout consists of only two elements (*EditorRegion IconPosition*). When one of the message-sending commands creates a new editor window, it chooses the first element of `LAFITE.EDITOR.LAYOUTS` that is not already in use. If all are in use, it prompts for a region.

LAFITE.EDITOR.SIZE [Variable]

Specifies the default size of message composition editors, other than those already specified in LAFITE.EDITOR.LAYOUTS. If the value of this variable is a size (*width* . *height*), then Lafite prompts for a region with a box this size. If the value is NIL, Lafite prompts for a general region, requiring you to drag from one corner to the opposite. The initial value is (470 . 300).

For backward compatibility, if LAFITE.BROWSER.LAYOUTS is NIL, it is considered to contain the single element (LIST LAFITEBROWSERREGION NIL LAFITEDISPLAYREGION). If LAFITE.EDITOR.LAYOUTS is NIL, it is considered to contain the single element (LIST LAFITEEDITORREGION). The variables LAFITEBROWSERREGION, LAFITEDISPLAYREGION, and LAFITEEDITORREGION are now obsolete and may be removed in a future release.

(LAFITE.BROWSE.FOLDER *foldername layout options* —) [Function]

Programmatic interface to the **Browse** command. Browses the mail folder named *foldername* and returns a mailfolder object. If a browser already exists for *foldername* it is returned; otherwise, a new browser is created. The normal Lafite defaulting rules apply to *foldername*, so, for example, "active" defaults to the file "active.mail" on the directory specified by LAFITEDEFAULTHOST&DIR. The argument *layout* is a list of up to three elements describing the layout of the browser; it takes the same form as elements of LAFITE.BROWSER.LAYOUTS. *options* is a list of zero or more keywords from among the following:

:ACTIVE	The browser is considered an "active" mail browser. Currently the only import of this setting is to enable GetMail on expansion (see below).
:GETMAIL	After loading the mail folder, retrieve new mail, if any, into this folder.
:SHRINK	After loading the mail folder, shrink the browser window. This option is ignored if :GETMAIL was specified.
:FORGET	If *foldername* is new to Lafite, don't remember it; i.e., do not add it to the menu of mail folders.
:CONFIRM	If *foldername* does not exist, require confirmation before creating it.

LAFITE.BROWSER.ICON.PREFERENCE [Variable]

Specifies where a browser's icon should be positioned if it has not already been specified in a browser layout as described above. The possible values are

NIL	Place the icon at the bottom left corner of the browser window. This is the default.
:ASK	Prompt for a position.
a function	Call this function, passing the browser window as its sole argument. The function should return a position, or NIL to make Lafite prompt for a position.

LAFITE.FOLDER.MENU.FONT [Variable]

The font used in the menu of folders displayed by the **Browse** and **MoveTo** commands. Initially NIL, which means use the default system MENUFONT.

`LAFITE.FOLDER.ICON` [Variable]

An icon specification for the icon used by Lafite browsers. It is of the form expected by `TITLEDICONW`, i.e., (*IconBitmap MaskBitmap TitleRegion*). This variable replaces the undocumented variables `MSGFOLDERTEMPLATE`, `MSGFOLDERICON`, and `MSGFOLDERMASK`.

`LAFITE.MSG.ICON` [Variable]

An icon specification for the icon used by Lafite message editors. This variable replaces the undocumented variables `MSGUNSENTREGION`, `MSGUNSENTICON`, and `MSGUNSENTMASK`.

The function `LAFITE` itself has been extended to take advantage of the same options available to `LAFITE.BROWSE.FOLDER`, so that you can set up your entire default mail screen in a single call. Here is the new documentation:

`(LAFITE` *on/off* `&optional` *folder* `&rest` *options*`)` [Function]

Turns Lafite on or off according to the first argument, which is one of the keywords `:ON`, `:OFF`, or `:RESTART`. For backward compatibility, the same symbols are accepted in the Interlisp package. `:ON` turns Lafite on, and has no effect if it is already on. `:OFF` turns it off, and has no effect if it is already off. `:RESTART` is equivalent to `(LAFITE :OFF)` followed by `(LAFITE :ON` *folder . options*`)`.

The argument *folder* specifies a mail folder to browse. If it is omitted, i.e., for a simple `(LAFITE :ON)`, it defaults to the value of `DEFAULTMAILFOLDERNAME`. `NIL` specifies no folder. So that you can specify *options* and still browse the default folder, *folder* = `T` also defaults to `DEFAULTMAILFOLDERNAME`.

The remaining arguments are optional keywords specifying what to do with the folder being browsed. The possible values are `:ACTIVE`, `:GETMAIL`, `:SHRINK`, `:FORGET`, and `:CONFIRM`, which are handled as described under `LAFITE.BROWSE.FOLDER`. For example, `(LAFITE :ON "active" :CONFIRM :SHRINK)` turns Lafite on, browses the folder "active", requiring confirmation if the file does not exist, and then shrinks the browser.

If *folder* is a list, it is interpreted as a list of specifications for files to browse and their placement. Each element is of the form (*foldername BrowserRegion IconPosition DisplayRegion . options*), which specifies invoking `(LAFITE.BROWSE.FOLDER` *foldername* `(LIST` *BrowserRegion IconPosition DisplayRegion*`)` *options*`)`. If there were any *options* specified in the call to `LAFITE` itself, they are appended to the *options* specified in each element of *folder*. See `LAFITE.BROWSE.FOLDER` for details.

Message Display

The *—End of message—* indication is back in operation for those of you who missed it in Medley.

Middle-button on the **Display** command now always prompts for its own display region (a box if `LAFITE.DISPLAY.SIZE` is non-`NIL`), and this window survives Updates; i.e., it stays open until you explicitly close it.

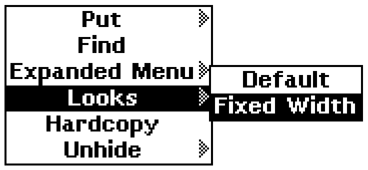
It is now possible to tell Lafite you do not care to see certain fields in the headers of messages, such as the numerous fields inserted by Arpanet mail transport.

`LAFITE.DONT.DISPLAY.HEADERS` [Variable]
`LAFITE.DONT.FORWARD.HEADERS` [Variable]
`LAFITE.DONT.HARDCOPY.HEADERS` [Variable]

The value of each of these variables is a list of strings specifying the names of the fields you wish omitted by the **Display**, **Forward**, and **Hardcopy** commands, respectively. The alphabetic case of the strings is unimportant. When fields are omitted from a displayed message, an asterisk appears in the titlebar of the message. You can see the omitted fields by using the **Unhide** command from the titlebar menu (see below). There is no corresponding command to see the omitted headers in a forwarded message or, of course, in a hardcopied message.

The value `("Return-Path" "Redistributed" "Received" "Message-Id")` is useful for filtering most of the rubbish from Arpanet headers.

Left- or Middle-button in the titlebar of a displayed message gives a modified TEdit menu of operations.



Those specific to Lafite are as follows:

Looks This command is used to change the appearance of the message. If you have selected a region of the message (longer than a single character), the command affects only that region; otherwise it applies to the whole message. This command affects only the display of this message this time; it has no effect on the message living in the mail folder.

The main **Looks** command brings up the same set of three menus as the normal TEdit Looks command, allowing you to choose any font and size. The subcommand **Fixed Width** changes the font to a fixed-width font, useful for viewing text formatted by folks in the fontless world. In addition, when changing to a fixed-width font, Lafite also sets the tab size to be the width of eight characters, something TEdit unfortunately doesn’t do automatically. The subcommand **Default** sets the font back to Lafite’s normal display font (the value of `LAFITEDISPLAYFONT`).

Hardcopy This command sends this single message to the default printer in whatever appearance it currently has. The hardcopy is sent immediately, independent of the setting of `LAFITEHARDCOPYBATCHFLG`.

Unhide If some fields of the message header have been omitted because they matched the strings in `LAFITE.DONT.DISPLAY.HEADERS`, this command reveals them. The subcommand **Hide** can be used to hide them again.

This menu can be further tailored using the following variables:

`LAFITE.EXTRA.DISPLAY.COMMANDS` [Variable]

This is the list of menu items added to the three possibly interesting TEdit commands offered in a message display window. Each item should be set up to return a function of one argument, which is applied to the message's text stream when the item is selected.

`LAFITE.LOOKS.SUBCOMMANDS` [Variable]

This is the list of menu subitems under the **Looks** command. It has the same format as `LAFITE.EXTRA.DISPLAY.COMMANDS`. At run time, Lafite inserts this list into the **Looks** item of `LAFITE.EXTRA.DISPLAY.COMMANDS`. The following function may be useful in implementing new Looks commands:

`(LAFITE.SET.LOOKS` *textstream newlooks paralooks*`)` [Function]

Changes the character looks of the current selection in *textstream*, or the whole stream if no more than one character is selected, to be *newlooks*, which can be a font descriptor or any argument acceptable to `TEDIT.LOOKS`. If *paralooks* is specified, it is a set of paragraph looks as expected by `TEDIT.PARALOOKS`. If *newlooks* is a fixed-width font and *paralooks* is `NIL`, this function also sets the default tab width of the selected text to be eight times the character width.

`LAFITEFIXEDWIDTHFONT` [Variable]

The font used by the **Fixed Width** command. It is initially the value of `DEFAULTFONT`, unless that font is variable-width, in which case it is set to Gacha 10.

`(LAFITE.HARDCOPY.TAB.WIDTH)` [Function]

When the **Fixed Width** command is used on a message, the tab width set for display is unlikely to be appropriate for hardcopy. If you subsequently issue the Hardcopy command on that window, Lafite has to guess what the tab width should be. Currently it does so by calling this function, whose initial definition computes a tab width based on `LAFITEFIXEDWIDTHFONT` coerced to Interpress. If you defaultly hardcopy to some other kind of printer, you may want to change this.

Browser Commands

Middle-button on the **Update** command bypasses the normal Update options menu. Its behavior is controlled by the following variable:

`LAFITE.MIDDLE.UPDATE` [Variable]

A list of keywords specifying the normal way by which you like to update browsers. The choices are:

`:UPDATE`	Write out changes, but do not expunge deleted messages.
`:EXPUNGE`	Expunge deleted messages. If there are no deleted messages, this is equivalent to `:UPDATE`. Both `:UPDATE` and `:EXPUNGE` will also process any pending hardcopy (when `LAFITEHARDCOPYBATCHFLG` is `T`).
`:SHRINK`	After updating the folder, shrink the browser.
`:CLOSE`	After updating the folder, close the browser.
`:CONFIRM`	Require confirmation (via mouse) of the operation.

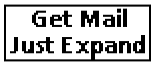
The keywords `:UPDATE` and `:EXPUNGE` are mutually exclusive, as are `:SHRINK` and `:CLOSE`. The default value is `(:EXPUNGE :SHRINK :CONFIRM)`, which means that middle-button on **Update** prompts with a message like "Click LEFT to confirm Expunge and Shrink". The message specifies the operation implied by the setting of the variable, rather than always being "Expunge"; e.g., it might read "Click LEFT to confirm Hardcopy, Update table of contents and Shrink". As a special case, when the folder needs no updating at all and `LAFITE.MIDDLE.UPDATE` specifies `:SHRINK`, Lafite shrinks the browser immediately without asking for confirmation.

The menu of Update options offered by the Update, Shrink, and Close commands now more accurately reflects what needs to be done. In particular, there is no Expunge option if you have not deleted any messages.

The widest labels on the browser menu (**Undelete** and **Hardcopy**) have been shortened. With Lafite's default menu font (Helvetica 10 Bold), this allows browsers to be 54 points narrower than before, so that two browsers can fit side by side on the screen. (I personally prefer Modern 12 Bold (10 for users of `NSDISPLAYSIZES`), which yields a browser another 18 points narrower.) You can (awkwardly) change the labels by editing `LAFITEBROWSERMENUITEMS`.

`LAFITEHARDCOPY.MIN.TOC` now works even when `LAFITEHARDCOPYBATCHFLG` = `T`. (If more than `LAFITEHARDCOPY.MIN.TOC` messages are being hardcopied, the output is preceded by a table of contents.)

If you have ever performed **GetMail** on a browser (or browsed it with the `:ACTIVE` option), then the behavior of middle-button on the browser's icon behaves differently: instead of just expanding the window, it pops up a menu offering to get mail.



Choosing **GetMail** is equivalent to expanding the window, then choosing the regular **GetMail** command, but of course requires less interaction on your part.

Lafite browsers now support copy selection. Selecting with the mouse in a browser window while holding down the Copy or Shift keys selects the text of the field you are pointing at. You can select the Date, From, or Subject field. Selecting to the left of the Date field selects the whole message, producing a "summary line" such as "#185 14 Jun From: Carstairs.pa -- Meeting rescheduled". This use of Shift for copy selection means that only the CTRL key can be used to remove messages from the selection, an operation previously supported by either CTRL or Shift.

The title field of the browser has been rearranged so that the most interesting information appears first, and is thus less likely to fall off the right edge of the window. For example, the browser for {FS}<Carstairs>Mail>Active.mail might have a title reading "Browsing Active (Move To: Memos) on {FS}<Carstairs>Mail>".

`LAFITEMOVETOCONFIRMFLG` [Variable]

This variable now only affects what happens when you choose the destination of a move from the menu (or using the middle-button accelerator), rather than typing it in (having selected "Other"). To summarize:

`NIL`	Do not require confirmation.
`LEFT`	Confirm only left-button MoveTo.
`MIDDLE`	Confirm only middle-button MoveTo.
`ALWAYS`	Require confirmation in all cases. This is the default.

Typing in a destination from the keyboard is enough evidence that you didn't slip on a mouse button, so confirmation is only required then if the destination file does not yet exist.

Mail Sending

You can now control whether Lafite bothers to ask you about whether to send a message formatted or not:

`LAFITE.SEND.FORMATTED` [Variable]

This variable is consulted when you ask to deliver a message that has what TEdit considers non-trivial formatting—font shifts, paragraph formatting, image bjects, or NS characters. The choices are:

`NIL`	Send it unformatted.
`:ASK`	Prompt with a menu.
`T`	Send it formatted without asking.

The value can also be a list of elements (*formattingType answer*), specifying that when `(TEDIT.FORMATTEDFILEP` *message*`)` is *formattingType* the decision is *answer*. The initial value is

```
((NSCHARS :ASK)
 (CHARLOOKS :ASK)
 (PARALOOKS :ASK)
 (IMAGEOBJ T))
```

meaning to send messages with image objects formatted without asking (sending them unformatted is always a bad idea), but ask for all other kinds of formatted messages.

You can now send unformatted messages containing NS characters. However, note that the Koto version of TEdit did not understand NS characters in unformatted text, so you should not do this if you expect that some of your recipients have not yet upgraded to Lyric. If you never send messages to Koto Lisp users (or don't care about this shortcoming), you might want to set the `NSCHARS` component of `LAFITE.SEND.FORMATTED` to `NIL`, i.e., have Lafite never send a message formatted merely because it contains NS characters.

Programs that compose message forms, such as those invoked from `LAFITESPECIALFORMS`, can specify that the message is to be delivered formatted without asking by giving the message (a textstream) the `TEXTPROP` property `LAFITEFORMAT`, value `TEDIT`.

Here are a couple more message customization variables:

`LAFITE.SIGNATURE` [Variable]

A string with which to "sign" your messages. This string is appended to the standard message forms—those produced by the **SendMail**, **Answer**, and **Forward** commands—always starting it on a new line. There should be a carriage return at the beginning of the string if you want a blank line to set off the signature. The initial value is `NIL`, meaning no signature.

`LAFITE.GV.FROM.FIELD` [Variable]

A string to use as the contents of the "From:" field of any Grapevine message you send that does not already have a From field. It must be a valid Grapevine address specification that parses down to your Grapevine identity, including registry. The usual syntax is "Human name <MailName.reg>", e.g., "Johann Amadeus Slonimsky III <Slo.pa>".

These two variables are examples of Lafite "personal" variables, in that they are closely bound to the logged-in user. The other is `LAFITEDEFAULTHOST&DIR`. If you change the logged in user (e.g., by calling `(LOGIN)`), Lafite resets all these variables to `NIL`, but remembers their values in case you log in again as yourself. This is designed to reduce confusion if you have more than one person using Lafite on the same machine. The values for each user are remembered as the value of the variable `LAFITE.USER.INFO`, an association list of user name and values, the latter being in property list format. Ordinarily you need not be aware of this variable, but you might, for example, want to initialize it with information about many users of a public machine.

Grapevine

Lafite now does a more thorough job of parsing Grapevine messages according to the RFC 822 specification and objecting to malformed addresses. It also preserves full addresses when composing replies, rather than discarding all the commentary.

Lafite now supports "private distribution list" syntax in Grapevine messages. A private distribution list (dl) is an indirection mechanism: it allows you to name a possibly large group of addressees without explicitly enumerating them in the header of the message. Unlike a public dl, whose membership is maintained in the Grapevine database, a private dl is simply a file in which you have stored the real names. The names in the file must be in standard Grapevine syntax, just as you would type in the header of a message, but with two exceptions: (1) Names may be separated with one or more carriage returns, rather than commas, to enhance readability; and (2) The names must be fully qualified, i.e., the Grapevine registry must be included. The latter requirement eliminates confusion in the case where the current sender is in a different registry than the maintainer of the dl. The names can make use of all the standard Grapevine syntax, which allows you to include comments in the file. For example, you could have a line reading

    <Carstairs.wbst>   "Works with bj on the ACME project"

The file may be a TEdit file; Lafite strips out the formatting before attempting to parse the addresses in it.

The syntax for a private dl in a message is

    To: *dlfilename*:;

i.e., the name of the file containing the real names, followed immediately by the two characters colon, semi-colon. If the file name contains any characters whose syntax is significant to Grapevine messages—colon, semi-colon, parentheses, brackets—the file name must be enclosed in double quotes. For example,

    To: "{FS9:PARC:Xerox}<Carstairs>Mail>Friends":;

The file name need not be complete; Lafite fills in defaults to locate the file according to the following two variables:

`LAFITEDL.EXT` [Variable]

The default extension of private distribution list files. If a private dl is specified without an explicit extension, Lafite uses this extension when looking for the file. The value is initially "DL".

`LAFITEDLDIRECTORIES` [Variable]

List of directories on which private distribution list files may be found. When you use a private dl in a message, Lafite searches LAFITEDEFAULTHOST&DIR, then this list, for the file containing the addressees. The value is initially NIL.

`*GV-SHOW-POSTMARK*` [Variable]

When Lafite retrieves Grapevine mail, there is additional information about the transport of each message that Lafite normally discards as uninteresting, as it is for the majority of messages. When the variable `*GV-SHOW-POSTMARK*` is true, Lafite preserves this information in an extra line in the header of the message. The line reports the authenticated name of the sender, the time at which the message entered the Grapevine system, and the host from which it entered; for example,

    GV-Info: Carstairs.pa at 4-Jun-88 23:38:00 from 204#16#

Note that the variable only affects the retrieval of mail, not its display. If the variable is NIL at the time of retrieval, the information is lost forever.

NS Mail

The NS mail retrieval code correctly interprets messages that are really returned messages. It now turns such messages into a description of a failed delivery, with a mail server as the apparent sender, instead of just putting a cryptic "TransportProblem" header in the message and leaving the header looking as though the message was from you.

The NS mail watcher no longer maintains a "session" on the mail server. This reduces server load a bit, and means that there is no confusion if another client (either Lafite or Viewpoint) tries to retrieve the mail of a user who happens to be running Lafite on another machine at the same time.

NS Mail now handles "attachments" better. This change was actually made in a special release of NS mail in September, 1987, but is included here for completeness, and to explain the extensions that handle reference objects. Attachments are no longer left in your mailbox to be read later with, for example, Viewpoint. Instead, Lafite retrieves the entire attachment and encapsulates it into an image object that is enclosed as part of the text message, immediately following the header. A typical attachment appears in a mail message as:

Attachment: Viewpoint Document

If you click inside the object with any mouse button, you are offered a menu of things you can do with the attachment. The choices vary according to the type of attachment:

View as text This brings up a window in which is displayed the raw content of the attachment as ascii bytes. Runs of non-ascii bytes are replaced by nulls to reduce the amount of garbage. Some attachments are utter gibberish, but some, such as Viewpoint documents and Interpress masters, contain sections that are plain text. With this command, you may be able to decide whether you care to do anything further with the attachment. (Sorry, there is no Viewpoint to TEdit converter, nor are there plans for one.)

Put to file This prompts you for a file name, and creates a file to contain the attachment. The file must be on an NS file server for this command to be very useful; otherwise, information will be lost. Once the file is so stored, you can retrieve it from Viewpoint and manipulate it just as if you had originally retrieved it as mail in Viewpoint. If you are running in Lyric, you must have the module NSRANDOM loaded for this command to work.

Send to Printer This command is only available for attachments that are in the form of an Interpress master. The command prompts you for a printer (which must be one that accepts Interpress, of course), and sends the attachment to it for printing.

Expand folder This command is only available for attachments that are in the form of a "folder". A folder is a mechanism for collecting several objects into a single one. The **Expand folder** command splits the attachment up into its component objects, each of which can be manipulated in the same way as a top-level attachment. For example, if the folder contains an Interpress master, you can print it.

If you use the **Put to file** command on a folder, the name component of the file name you type will be treated as the name of a new subdirectory, and the components of the folder will appear as files in that subdirectory. For other types of attachments, **Put to file** (usually) produces an ordinary (non-directory) file.

If the attachment is a Viewpoint "Reference" object, the **View as text** and **Send to Printer** commands apply to the file the reference mentions, not the attachment itself. The **Put to file** command is renamed **Store reference**, to make it clear it applies to the reference object itself, rather than the file being referred to (which there is no point in copying, since Viewpoint can deal with the reference object itself just fine). References to folders also offer the command **FileBrowse**, which brings up a FileBrowser (if you have it loaded) browsing the directory referred to.

Messages containing attachments are otherwise just like formatted messages—you can move them to other folders, and you can forward them (assuming the mail is received by another Lafite recipient and did not have to pass through an information-losing mail gateway).

There is currently no mechanism for creating your own attachents to send to other users.

Miscellaneous

The middle-button **Browse** command has a new subcommand **Browse & Forget**. This command prompts for a file name and browses it with the `:FORGET` option, i.e., it does not add it to the folder menu. Note that this command does not remove an existing folder from the menu, it merely refrains from adding a new one.

Lafite's table of contents code now handles messages containing NS strings in the Subject field. This fixes the bug where browsing such a file would claim that the toc was inconsistent with the folder, throw away the toc file and reparse from scratch. Also, the bug where Lafite reported "Folder is Empty" when discarding a malformed toc has been fixed.

Lafite now treats folder names as strings, and does its best to (a) ignore alphabetic case within folder names and (b) preserve the case that you type when creating a folder. This means, among other things, that you can now have mixed case in your `LAFITEDEFAULTHOST&DIR`, and that you can have folder names that are entirely numeric, such as "1186".

Lafite also keeps in its folder menu what the server considers the "canonical" name of each file to be. This means that if your folders include a file not on your default directory with a host not specified by its canonical name, then the next time you browse this folder, Lafite will think it is a new folder, leaving you with a menu containing both the old and the new names. Use the **Forget Folders** subcommand of **Browse** to dispose of such duplicates.

The information about your folders and message forms, formerly maintained in a file named LAFITE.PROFILE on your `LAFITEDEFAULTHOST&DIR`, is now kept in a file named LAFITE.INFO in a more extensible format. When you first run the new Lafite, if you have no LAFITE.INFO file, it will read LAFITE.PROFILE instead and convert to the new format.

Lafite does a better job now of noticing whether a mail file has been changed out from under you. It checks every time it opens the file, not just after logout. If the mail file has been changed but you have unsaved changes to the browser, Lafite will offer to save your changes if the mail file has merely been appended to. If, however, it has been expunged, so that the messages all live in different locations in the file, this is not possible, and the changes in your browser must be discarded.

When you return from logout and Lafite can't locate the file behind one or more of your browsers, it no longer closes the browser. The next time you try to do anything with the browser, such as display a message, Lafite will again try to locate the file. Thus, if the file server holding the file is down, you need only wait for it to come back up before using the browser, rather than having to rebrowse the folder.

The after-logout code now runs mostly in a separate process, which allows the call to LOGOUT to return faster.

Lafite files are now of type LAFITE, rather than TEXT, since they typically contain non-text content, such as NS characters and TEdit formatting. On any kind of server but an NS file server, type LAFITE is usually coerced to BINARY.

Changes to Programmer Interface

Well, actually, there is no documented programmer's interface. Numerous internals have changed; this section lists but a few.

With the new handling of modes, nearly everything surrounding the use of \LAFITEUSERDATA and the record LAFITEOPS (and hence the variable LAFITEMODELST) has changed. POLLNEWMAIL has been restructured; PRINTLAFITESTATUS behaves differently. There is a new hook LAFITENEWMAILFN. When this variable is non-NIL, its value is called, with no arguments, when the Lafite status window notes new mail.

The MAILFOLDER record has changed slightly. The use of its FOLDERDISPLAYWINDOWS field is different. The new field FOLDERDISPLAYREGIONS replaces BROWSERSELECTIONREGION.

The whole family of functions around \LAFITE.BROWSE has substantially changed.

Several functions have been deleted or renamed including LA.REMOVEDUPLICATES, LA.SETDIFFERENCE, PROFILEFILENAME, \LAFITE.MERGE.PROFILES (now \LAFITE.MERGE.NAMELISTS), CHANGEFLAGINFOLDER, \LAFITE.GETMAILFOLDER.

Partial List of Fixed or Obsolete ARs

267	Easy font changing
307	Filter selected fields out of displayed msg?
552	Make it easier to change logged in user
1239	Recover from "Lafite is confused..."
1249	Want automatic GetMail on icon expansion
1273	Prompt for GV login if BadPassword error => Declined: use Login
1389	Answer command should preserve entire address
1644	Private dl's
2143	Want detachable display windows that survive Update
2421	Preserve case when creating files
2433	Use `GETBOXREGION` to place window?
2597	Check that GV headers conform to RFC822
3168	Want user-specifiable browser/menu layout
3174	List of regions for browser, display, etc
3269	Update, GetMail etc should check creationdate/eof and rebrowse if needed
4420	`LAFITEHARDCOPY.MIN.TOC` doesn't work with `LAFITEHARDCOPYBATCHFLG`
4628	addresses lacking a closing double quote break with illegal arg 65536

5109	Don’t loop if bad NS password
5110	Blank name (e.g. in Reply-to) completes to *:domain:org
5111	Want name parser that doesn’t add default registries
5116	Parser loops forever if header address has unmatched paren or quote
5117	Want way of dealing with NS Mail attachments without running Star
5121	Close/Shrink menu includes "Expunge" even if no deleted messages
5122	Simultaneously watch for both NS and GV mail
5125	Case sensitivity in `LAFITEDEFAULTHOST&DIR` screws up MoveTo
5129	Want browser title rearranged to avoid clipping important info
5134	Message from self not noted if user not authenticated when folder browsed
5137	Want to be able to keep display window open after browser shrunk
5300	Answer NS message in GV mode does not give To field
5541	Default status window off screen on small 1186 screen
5638	Prune headers on forwarded messages
6372	`LAFITENEWMAILTUNE` not on daybreak
6260	Blank To/cc line parsed incorrectly
6547	Can’t put Arpa address in From field
6918	Send unformatted dies in `COPYBYTES` illegal arg
7244	Numeric folder names die in string-equal
8643	`(il:lafite 'il:off)` breaks in `SHADEITEM`, menu = `nil`
8679	Changing modes with Lafite off doesn’t reset `\LAFITEUSERDATA`?
8995	When toc discarded, says "Folder empty"

LAFITE UPDATE

Last Edited: November 16, 1989

This document describes changes to Lafite since the October, 1986 release. Comments and suggestions are welcome on the new features described herein.

Installation

To install Lafite in a sysout where Lafite has never been loaded, simply load LAFITE.LCOM from the directory appropriate to your version of Lisp. However, if an older version of Lafite is already loaded in your sysout, you must instead Quit out of the old Lafite, and explicitly load *all* of Lafite's component files. If the old version of Lafite in your sysout is dated June, 1988 or later, you can do this by calling (LOAD-LAFITE "*directory*"), after which you may still need to load MAILSCAVENGE.LCOM if the old version was in your sysout; otherwise, load *all* of the following LCOMs from the appropriate directory:

LAFITEBROWSE	NSMAIL
LAFITEMAIL	MAILSCAVENGE
LAFITESEND	LAFITEFIND
MAILCLIENT	LAFITE

Of the files MAILCLIENT, NSMAIL, MAILSCAVENGE, LAFITEFIND, you need only load those that are already present in your sysout from the old Lafite (all are present in the standard Parc sysout, so all must be reloaded). Note that LAFITE.LCOM must be loaded last.

Compatibility

This new Lafite is incompatible with most modules that thought they knew something about the internals of Lafite. Specific cases (this is mostly old news by now):

Rooms. Rooms has been updated to know about the new Lafite.

Lens. Lens has been updated to know about the new Lafite.

LafitePrivateDL. This LispUsers module is incompatible with the new Lafite, and also obsoleted by it (see the section "Grapevine" below).

LafiteHighlight. This probably still works, but you may prefer to use header filters in the new Lafite instead (see "Message Display" below).

LafiteTimedDelete. This probably still works.

Vanilla Init. Stan says he now has this working properly with the new Lafite.

DateSort. This module is obsoleted by the new Lafite.

Lafite-Indent. This is actually a TEdit macro package and is independent of Lafite.

Lafite Modes

Lafite is now willing to operate in more than one mode "simultaneously". The mail watch background task checks all of your mailboxes in all the modes that are active, and **GetMail** retrieves mail from all mailboxes**.** The variable `LAFITE.USE.ALL.MODES` controls whether Lafite runs in more than one mode at once. The interesting values are:

`NIL`	This is the old way, where you must change modes manually. All commands operate only in the currently active mode.
`:POLL`	The mail watch task checks all modes, but **GetMail** only retrieves in the currently active mode.
`T`	Mail watch checks and **GetMail** retrieves in all modes.

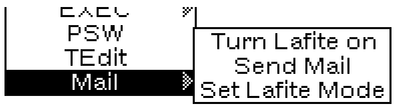
The initial value is `T`. The flag has no effect on the **SendMail** command, which continues to bring up a message composition window in the current mode. It also does not affect the **Answer** command, which answers in the mode that the message was retrieved in. The retrieval mode is remembered in the table of contents file; in the absence of this knowledge, e.g., if the toc is deleted or the message was retrieved with an older Lafite, the **Answer** command uses simple rules that nearly always unambiguously distinguish GV from NS messages. Be aware that the value `:POLL` can be confusing to such hacks as automatic mail retrievers, since the status window can report new mail, yet **GetMail** won't retrieve it.

Ordinarily, Lafite only attempts to authenticate you in the non-primary modes when it starts up, and after logout. If, for example, you are in GV mode, and the NS authentication service is down when you start up Lafite, or your NS password is incorrect, it does not repeatedly try to authenticate you, but merely reports the condition once and behaves as if the failing mode does not exist. If conditions later change, you can force another attempt by using the mode switching commands to explicitly switch to the mode. If your password is incorrect, you can change it using the NS Login subcommand of Quit (see below).

GetMail only attempts to retrieve mail from the servers that reported new mail in the most recent poll. This means that if one of your mail servers is not responding, you no longer have to wait during **GetMail** for it to time out. If you click **GetMail** at a time when the status window reports "No New Mail", then **GetMail** will attempt to contact all of your mailbox servers, just as it did in previous releases.

Turning Lafite on and off

You can now turn Lafite on or switch modes from the background menu.



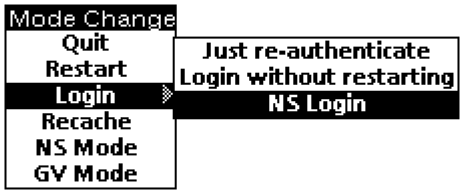
Mail This brings up a standard message sending window in the current mode, whether Lafite is on or not. If Lafite is not on, it will attempt to authenticate you in the current mode.

Send Mail This is the same as the main **Mail** command, except that you get to choose which kind of message form, just as middle-button on the Lafite status window's **Send Mail** command allows.

Turn Lafite On This is the same as calling `(LAFITE :ON)`. It has no effect if Lafite is already on.

Set Lafite Mode This command allows you to change Lafite's mode. It offers a menu of the mode choices found also as part of the middle-button **Quit** command on the Lafite status window.

The middle-button **Quit** command on the Lafite status window has two new subcommands, **Recache** and **Login**. Here, for completeness, is a description of all the commands:



Quit This is the same as the main command, which is the same as calling `(LAFITE :OFF)`. It closes all folders, asking about updating them if any need it, then turns off Lafite.

Restart This turns Lafite off, then immediately back on; same as `(LAFITE :RESTART)`.

Login Changes the logged-in user for Lafite. This command first turns Lafite off, so that any updating required by closing out the previous user's folders happens under the previous user's login. It then prompts for user name and password, exactly as if you had called `(LOGIN)` yourself, and then for a new `LAFITEDEFAULTHOST&DIR`. Finally, it turns Lafite back on.

Just re-authenticate Re-authenticates the currently logged-in user without prompting for a new login. This may be useful if Lafite's original attempt at authenticating the user failed or got inconsistent information due to network difficulties.

Login without restarting Performs a login without changing the state of Lafite. This is useful if you had previously logged in with an incorrect password, or if you have multiple user identities all sharing the same set of folders.

NS Login Prompts for a new login just for NS mail. You may need to use this if your global user name does not correspond to your NS user name or an alias of it in the default domain, or if you have different passwords for Grapevine and NS. There is actually one of these commands for each mode that understands an independent login, but NS is the only current instantiation.

Recache In the case of several of the public interface variables, Lafite computes other structures based on them, and does not automatically recompute them when you change the public variable. The recache command forces Lafite to recompute these "caches" of the variables' values. A recache also occurs whenever Lafite is (re)started. The command is implemented by the function `LAFITE.COMPUTE.CACHED.VARS`, which you can advise if you have other caches that Lafite doesn't know about. The variables currently affected by this command (i.e., changes in whose values are not otherwise noticed) are:

```
LAFITE.DONT.DISPLAY.HEADERS
LAFITE.DONT.FORWARD.HEADERS
LAFITE.DONT.HARDCOPY.HEADERS
LAFITE.EXTRA.DISPLAY.COMMANDS
LAFITE.LOOKS.SUBCOMMANDS
LAFITE.GV.FROM.FIELD
LAFITE.HOST.ABBREVS.
```

NS Mode
GV Mode Switch to the specified mode. There is one of these for each mode.

Screen Appearance

There are several new variables and one function for controlling the appearance of your Lafite screen.

`LAFITE.BROWSER.LAYOUTS` [Variable]

A list of browser "layouts" specifying where Lafite should place browsers, their display windows and their icons. Each layout is of the form (*BrowserRegion IconPosition DisplayRegion*). *IconPosition* and *DisplayRegion* are optional. The region variables are standard window system regions of the form (*left bottom width height*); *IconPosition* is a position of the form (*x* . *y*). When the **Browse** command creates a new browser window, it chooses the first element of `LAFITE.BROWSER.LAYOUTS` that is not already in use. If all are in use, it prompts for a region.

`LAFITE.DISPLAY.SIZE` [Variable]

Specifies the default size of display windows, other than those already specified in a browser layout. If the value of this variable is a size (*width* . *height*), then Lafite prompts for a region with a box this size. If the value is `NIL`, Lafite prompts for a general region, requiring you to drag from one corner to the opposite. The initial value is `(500 . 300)`.

`LAFITE.EDITOR.LAYOUTS` [Variable]

A list of editor "layouts" specifying where Lafite should place message composition editors and their icons. As message editors have only a single window, each layout consists of only two elements (*EditorRegion IconPosition*). When one of the message-sending commands creates a new editor window, it chooses the first element of `LAFITE.EDITOR.LAYOUTS` that is not already in use. If all are in use, it prompts for a region.

`LAFITE.EDITOR.SIZE` [Variable]

Specifies the default size of message composition editors, other than those already specified in `LAFITE.EDITOR.LAYOUTS`. If the value of this variable is a size (*width* . *height*), then Lafite prompts for a region with a box this size. If the value is `NIL`, Lafite prompts for a general region, requiring you to drag from one corner to the opposite. The initial value is `(470 . 300)`.

For backward compatibility, if LAFITE.BROWSER.LAYOUTS is NIL, it is considered to contain the single element (LIST LAFITEBROWSERREGION NIL LAFITEDISPLAYREGION). If LAFITE.EDITOR.LAYOUTS is NIL, it is considered to contain the single element (LIST LAFITEEDITORREGION). The variables LAFITEBROWSERREGION, LAFITEDISPLAYREGION, and LAFITEEDITORREGION are now obsolete and may be removed in a future release.

(LAFITE.BROWSE.FOLDER *foldername layout options* —) [Function]

Programmatic interface to the **Browse** command. Browses the mail folder named *foldername* and returns a mailfolder object, or NIL on failure. If a browser already exists for *foldername* it is returned; otherwise, a new browser is created. The normal Lafite defaulting rules apply to *foldername*, so, for example, "active" defaults to the file "active.mail" on the directory specified by LAFITEDEFAULTHOST&DIR. The argument *layout* is a list of up to three elements describing the layout of the browser; it takes the same form as elements of LAFITE.BROWSER.LAYOUTS. *options* is a list of zero or more keywords from among the following:

:ACTIVE	The browser is considered an "active" mail browser. Currently the only import of this setting is to enable GetMail on expansion (see below).
:GETMAIL	After loading the mail folder, retrieve new mail, if any, into this folder.
:SHRINK	After loading the mail folder, shrink the browser window. This option is ignored if :GETMAIL was specified.
:FORGET	If *foldername* is new to Lafite, don't remember it; i.e., do not add it to the menu of mail folders.
:CONFIRM	If *foldername* does not exist, require confirmation before creating it.
:OLD	If *foldername* does not exist, do not create it, but return NIL instead.

LAFITE.BROWSER.ICON.PREFERENCE [Variable]

Specifies where a browser's icon should be positioned if it has not already been specified in a browser layout as described above. The possible values are

NIL	Place the icon at the bottom left corner of the browser window. This is the default.
:ASK	Prompt for a position.
a function	Call this function, passing the browser window as its sole argument. The function should return a position, or NIL to make Lafite prompt for a position.

LAFITE.FOLDER.MENU.FONT [Variable]

The font used in the menu of folders displayed by the **Browse** and **MoveTo** commands. Initially NIL, which means use the default system MENUFONT.

LAFITE.FOLDER.ICON [Variable]

An icon specification for the icon used by Lafite browsers. It is of the form expected by TITLEDICONW, i.e., (*IconBitmap MaskBitmap TitleRegion*). This variable replaces the undocumented variables MSGFOLDERTEMPLATE, MSGFOLDERICON, and MSGFOLDERMASK.

`LAFITE.MSG.ICON` [Variable]

An icon specification for the icon used by Lafite message editors. This variable replaces the undocumented variables `MSGUNSENTREGION`, `MSGUNSENTICON`, and `MSGUNSENTMASK`.

The function `LAFITE` itself has been extended to take advantage of the same options available to `LAFITE.BROWSE.FOLDER`, so that you can set up your entire default mail screen in a single call. Here is the new documentation:

`(LAFITE` *on/off* `&optional` *folder* `&rest` *options*`)` [Function]

Turns Lafite on or off according to the first argument, which is one of the keywords `:ON`, `:OFF`, or `:RESTART`. For backward compatibility, the same symbols are accepted in the Interlisp package. `:ON` turns Lafite on, and has no effect if it is already on. `:OFF` turns it off, and has no effect if it is already off. `:RESTART` is equivalent to `(LAFITE :OFF)` followed by `(LAFITE :ON` *folder . options*`)`.

The argument *folder* specifies a mail folder to browse. If it is omitted, i.e., for a simple `(LAFITE :ON)`, it defaults to the value of `DEFAULTMAILFOLDERNAME`. `NIL` specifies no folder. So that you can specify *options* and still browse the default folder, *folder* `= T` also defaults to `DEFAULTMAILFOLDERNAME`.

The remaining arguments are optional keywords specifying what to do with the folder being browsed. The possible values are `:ACTIVE`, `:GETMAIL`, `:SHRINK`, `:FORGET`, and `:CONFIRM`, which are handled as described under `LAFITE.BROWSE.FOLDER`. For example, `(LAFITE :ON "active" :CONFIRM :SHRINK)` turns Lafite on, browses the folder "active", requiring confirmation if the file does not exist, and then shrinks the browser.

If *folder* is a list, it is interpreted as a list of specifications for files to browse and their placement. Each element is of the form (*foldername BrowserRegion IconPosition DisplayRegion . options*), which specifies invoking `(LAFITE.BROWSE.FOLDER` *foldername* `(LIST` *BrowserRegion IconPosition DisplayRegion*`)` *options*`)`. If there were any *options* specified in the call to `LAFITE` itself, they are appended to the *options* specified in each element of *folder*. See `LAFITE.BROWSE.FOLDER` for details.

Message Display

The —*End of message*— indication is back in operation for those of you who missed it in early versions of Medley.

Middle-button on the **Display** command now always prompts for its own display region (a box if `LAFITE.DISPLAY.SIZE` is non-`NIL`), and this window survives Updates; i.e., it stays open until you explicitly close it.

It is now possible to tell Lafite you do not care to see certain fields in the headers of messages, such as the numerous fields inserted by Arpanet mail transport.

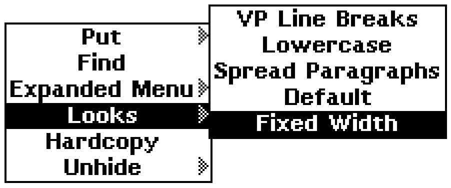
LAFITE.DONT.DISPLAY.HEADERS [Variable]
LAFITE.DONT.FORWARD.HEADERS [Variable]
LAFITE.DONT.HARDCOPY.HEADERS [Variable]

The value of each of these variables is a list of strings specifying the names of the fields you wish omitted by the **Display**, **Forward**, and **Hardcopy** commands, respectively. The alphabetic case of the strings is unimportant. When fields are omitted from a displayed message, an asterisk appears in the titlebar of the message. You can see the omitted fields by using the **Unhide** command from the titlebar menu (see below). There is no corresponding command to see the omitted headers in a forwarded message or, of course, in a hardcopied message.

The value `("Return-Path" "Redistributed" "Received" "Message-Id" "Errors-To")` is useful for filtering most of the rubbish from Arpanet headers.

If you include the symbol GV in the list, Lafite filters the ugly "GVGV..." supplemental header from NS messages that arrived via the GV gateway. This supplement includes any headers that didn't translate into NS land, such as comments in addresses ("Fred Carstairs <fc@foo.bar.com>") and all those "Received:" lines in arpa mail. By filtering this entire section you could occasionally miss an interesting header hiding in there, but you can still use **Unhide**, of course.

Left- or Middle-button in the titlebar of a displayed message gives a modified TEdit menu of operations.



Those specific to Lafite are as follows:

Looks This command is used to change the appearance of the message. If you have selected a region of the message (longer than a single character), the command affects only that region; otherwise it applies to the whole message. This command affects only the display of this message this time; it has no effect on the message living in the mail folder.

The main **Looks** command brings up the same set of three menus as the normal TEdit Looks command, allowing you to choose any font and size. The subcommands apply some built-in looks or other transformations:

Fixed Width Changes the font to a fixed-width font, useful for viewing text formatted by folks in the fontless world. In addition, when changing to a fixed-width font, Lafite also sets the tab size to be the width of eight characters, something TEdit unfortunately doesn't do automatically.

Default Sets the font back to Lafite's normal display font (the value of LAFITEDISPLAYFONT).

Spread Paragraphs Inserts 10-point leading between paragraphs. This is useful for messages that come from Tioga, which doesn't put a blank line between paragraphs in the plain text body.

Lowercase Converts message body to lowercase. This is useful for messages from those bozos who haven't discovered their Shift Lock yet, or are typing on Model 33 Teletypes. This and the next command change only the body of the message, not the header.

VP Line Breaks This turns the character #o35, which at least some mail senders in the XNS world seem to use as an end-of-line character, into carriage return.

Hardcopy This command sends this single message to the default printer in whatever appearance it currently has. The hardcopy is sent immediately, independent of the setting of `LAFITEHARDCOPYBATCHFLG`.

Unhide If some fields of the message header have been omitted because they matched the strings in `LAFITE.DONT.DISPLAY.HEADERS`, this command reveals them. The subcommand **Hide** can be used to hide them again.

This menu can be further tailored using the following variables:

`LAFITE.EXTRA.DISPLAY.COMMANDS` [Variable]

This is the list of menu items added to the three possibly interesting TEdit commands offered in a message display window. Each item should be set up to return a function of one argument, which is applied to the message's text stream when the item is selected.

`LAFITE.LOOKS.SUBCOMMANDS` [Variable]

This is the list of menu subitems under the **Looks** command. It has the same format as `LAFITE.EXTRA.DISPLAY.COMMANDS`. At run time, Lafite inserts this list into the **Looks** item of `LAFITE.EXTRA.DISPLAY.COMMANDS`. The following function may be useful in implementing new Looks commands:

`(LAFITE.SET.LOOKS` *textstream newlooks paralooks*`)` [Function]

Changes the character looks of the current selection in *textstream*, or the whole stream if no more than one character is selected, to be *newlooks*, which can be a font descriptor or any argument acceptable to `TEDIT.LOOKS`. If *paralooks* is specified, it is a set of paragraph looks as expected by `TEDIT.PARALOOKS`. If *newlooks* is a fixed-width font and *paralooks* is `NIL`, this function also sets the default tab width of the selected text to be eight times the character width.

`LAFITEFIXEDWIDTHFONT` [Variable]

The font used by the **Fixed Width** command. It is initially the value of `DEFAULTFONT`, unless that font is variable-width, in which case it is set to Gacha 10.

If you have in older versions of Lafite set `LAFITEDISPLAYFONT` to a fixed-width font just so you can easily read messages from outside users, you might consider setting `LAFITEDISPLAYFONT` back to a more pleasing font and using the **Fixed Width** command in the few cases where you need it.

`(LAFITE.HARDCOPY.TAB.WIDTH)` [Function]

When the **Fixed Width** command is used on a message, the tab width set for display is unlikely to be appropriate for hardcopy. If you subsequently issue the Hardcopy command on that window, Lafite has to guess what the tab width should be. Currently it does so by calling this function, whose initial definition computes a tab width based on `LAFITEFIXEDWIDTHFONT` coerced to Interpress. If you defaultly hardcopy to some other kind of printer, you may want to change this.

Browser Commands

Middle-button on the **Update** command bypasses the normal Update options menu. Its behavior is controlled by the following variable:

`LAFITE.MIDDLE.UPDATE` [Variable]

A list of keywords specifying the normal way by which you like to update browsers. The choices are:

`:UPDATE`	Write out changes, but do not expunge deleted messages.
`:EXPUNGE`	Expunge deleted messages. If there are no deleted messages, this is equivalent to `:UPDATE`. Both `:UPDATE` and `:EXPUNGE` will also process any pending hardcopy (when `LAFITEHARDCOPYBATCHFLG` is `T`).
`:SHRINK`	After updating the folder, shrink the browser.
`:CLOSE`	After updating the folder, close the browser.
`:CONFIRM`	Require confirmation (via mouse) of the operation.

The keywords `:UPDATE` and `:EXPUNGE` are mutually exclusive, as are `:SHRINK` and `:CLOSE`. The default value is `(:EXPUNGE :SHRINK :CONFIRM)`, which means that middle-button on **Update** prompts with a message like "Click LEFT to confirm Expunge and Shrink". The message specifies the operation implied by the setting of the variable, rather than always being "Expunge"; e.g., it might read "Click LEFT to confirm Hardcopy, Update table of contents and Shrink". As a special case, when the folder needs no updating at all and `LAFITE.MIDDLE.UPDATE` specifies `:SHRINK`, Lafite shrinks the browser immediately without asking for confirmation.

The menu of Update options offered by the Update, Shrink, and Close commands now more accurately reflects what needs to be done. In particular, there is no Expunge option if you have not deleted any messages.

The widest labels on the browser menu (**Undelete** and **Hardcopy**) have been shortened. With Lafite's default menu font (Helvetica 10 Bold), this allows browsers to be 54 points narrower than before, so that two browsers can fit side by side on the screen. (I personally prefer Modern 12 Bold (10 for users of `NSDISPLAYSIZES`), which yields a browser another 18 points narrower.) You can (awkwardly) change the labels by editing `LAFITEBROWSERMENUITEMS`.

LAFITEHARDCOPY.MIN.TOC now works even when LAFITEHARDCOPYBATCHFLG = T. (If more than LAFITEHARDCOPY.MIN.TOC messages are being hardcopied, the output is preceded by a table of contents.)

If you have ever performed **GetMail** on a browser (or browsed it with the :ACTIVE option), then the behavior of middle-button on the browser's icon behaves differently: if you hold down the button, it pops up a menu offering to get mail.

Choosing **GetMail** is equivalent to expanding the window, then choosing the regular **GetMail** command, but of course requires less interaction on your part. If you release the button without selecting **GetMail**, the browser is expanded, just as if there had been no menu interaction at all.

Lafite browsers now support copy selection. Selecting with the mouse in a browser window while holding down the Copy or Shift keys selects the text of the field you are pointing at. You can select the Date, From, or Subject field. Selecting to the left of the Date field selects the whole message, producing a "summary line" such as "#185 14 Jun From: Carstairs.pa -- Meeting rescheduled". This use of Shift for copy selection means that only the CTRL key can be used to remove messages from the selection, an operation previously supported by either CTRL or Shift.

The title field of the browser has been rearranged so that the most interesting information appears first, and is thus less likely to fall off the right edge of the window. For example, the browser for {FS}<Carstairs>Mail>Active.mail might have a title reading "Browsing Active (Move To: Memos) on {FS}<Carstairs>Mail>".

LAFITEMOVETOCONFIRMFLG [Variable]

This variable now only affects what happens when you choose the destination of a move from the menu (or using the middle-button accelerator), rather than typing it in (having selected "Other"). To summarize:

NIL	Do not require confirmation.
LEFT	Confirm only left-button MoveTo.
MIDDLE	Confirm only middle-button MoveTo.
ALWAYS	Require confirmation in all cases. This is the default.

Typing in a destination from the keyboard is enough evidence that you didn't slip on a mouse button, so confirmation is only required then if the destination file does not yet exist.

LAFITE.AUTO.MOVE.MENU [Variable]

Setting this variable non-NIL enables an automatic "accelerated Move To" menu. The menu is attached to your browser window and contains the names of all folders to which you have moved messages from this browser. Selecting a folder name with the mouse is equivalent to selecting the **MoveTo** command in the browser and choosing that folder name, with no confirmation required. The menu also contain the items "Display" and "Delete", which are simply copies of the normal browser commands. This option is convenient when walking through a browser, dispatching messages into other folders. Of course, to move a message into some other folder not in the menu, use the regular **MoveTo** command; afterwards, the folder you choose will be added to the accelerated menu.

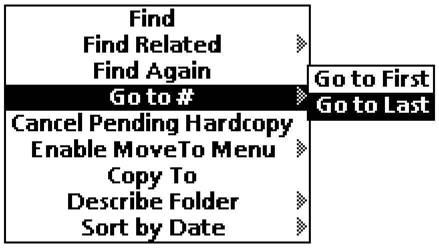
The menu does not appear until the first time you issue a **MoveTo** from this browser to a folder other than the default MoveTo (the one that middle-button chooses). It is attached by default to the right-hand side of the browser window, but you can set `LAFITE.AUTO.MOVE.MENU` to one of the symbols `LEFT`, `RIGHT`, `TOP` or `BOTTOM` to be explicit about where you want it.

You can get an accelerated Move To menu when `LAFITE.AUTO.MOVE.MENU` is NIL (or before you have moved to a second folder) by selecting **Enable Move To Menu** from the middle-button title-bar menu. In this case, you are asked to choose which folders you want in the menu.

You can separately close the menu without closing the browser window. To reopen the menu, use the subcommand **Restore Move To Menu** from the title-bar menu.

Browser Title-Bar Menu

Lafite now automatically loads the formerly separate module LAFITEFIND. In addition to the three Find commands available from the menu you get when you hold down the middle button in a browser's title bar, there are several new commands. All are described here briefly for completeness.



Find Searches the browser for one or more messages matching some search criteria, which are obtained by a series of additional menus and type-in:

1. **Find** {**Next** | **Previous**} {**One** | **All**} chooses between searching forward from the current message or backward, and whether to stop at the first matching message, or to go ahead and find all matching messages at once.

2. {**From** | **Subject** | **Body** | **Mark** | **Related**} indicates what part of the message to look at. **From** and **Subject** specify the corresponding fields in the header of a message; **Body** specifies the entire message; **Mark** specifies the single-character mark to the left of the message number in the browser window; **Related** specifies using the **Find Related** command (below).

Actually, the **From** search specification is intended to match what you see in the browser window, which means you can also use it to search for messages you sent *to* another person; i.e., **From** searches the From field of every message, and in addition searches the To field of messages from you.

A **Body** search is substantially slower than the others, since it has to scan the mail file, rather than just the table of contents.

3. Finally, you are prompted (in all except **Related**) to type a string (or single character if **Mark**) to search for. Alphabetic case is irrelevant.

On a successful search, Lafite leaves as the browser's selection the single matching message (for **One**) or all matching messages (for **All**).

Find Related Searches for all the messages "related" to this one; specifically, for any message whose subject contains the current message's subject (minus an initial "Re:", if any) as a substring. In most cases, this finds exactly the messages that were sent using the **Answer** command in reply to this message or to a subsequent reply. Of course, if the current message's subject is empty or some short uninformative phrase (e.g., "bug"), the search may find more messages than you intended. The search is normally forward; you can search backward with a subcommand. At the end of this command, all related messages, including the current one, are selected, so you can use Display to cycle through them, MoveTo to put them all in some other folder, etc. If you want to find only the *first* related message, use the **Find** command and choose the **Related** "part".

Find Again Repeats the previous **Find** command. This is most useful when the previous search was a **Find Next One** or **Find Previous One**, in which case you can step through all matching messages one at a time using this command. The "previous search" is global, i.e., not confined to the browser in which it was issued, so you can, for example, search one browser, then use **Find Again** to perform the same search in a different browser.

Go to # Scrolls the browser to a message specified by number and selects it (and unselects any other message). You specify the number either via mouse with the numberpad reader or by typing digits from the keyboard. The subcommands **Go to First** and **Go to Last** allow you to jump directly to the first or last message in the browser.

Cancel Pending Hardcopy If you have `LAFITEHARDCOPYBATCHFLG` set to `T` and have issued the Hardcopy command for one or more messages but not yet issued an Update (which will actually perform the hardcopy formatting), this command will cancel the hardcopy.

Enable Move To Menu Brings up an accelerated MoveTo menu, as described in the previous section. The subcommand **Restore MoveTo Menu** simply restores an earlier MoveTo menu that you had closed, without asking what folders should be in it.

Copy To Copies the selected message(s) to the folder of your choice. This is the same as **MoveTo** from the main menu, but it does not delete the message or mark it moved, nor is there a middle-button accelerator.

Describe Folder Prints in the browser's prompt window some descriptive information about the folder: its full file name, the number of messages, the number of deleted messages (if any), and the number of disk (512-byte) pages it occupies. The subcommand **Inspect Folder** is intended for debugging; it brings up an inspector on the browser's MAILFOLDER object.

Sort by Date Sorts the browser's contents by date. The subcommand **Sort Selected Range** sorts only between the first and last selected messages. These commands are described further below.

Sorting Mail

Lafite now has the ability to sort messages in a browser in order of their "Date:" fields, using the titlebar command **Sort by Date** or its subcommand **Sort Selected Range**. Messages lacking a Date field or having an unparseable Date field are sorted so as to remain next to the preceding message. The messages are sorted in the browser only, until you next **Update**, at which time the messages are written to the file in the sorted order. When messages have been rearranged, the **Update** option **Write Out Changes Only** is not available, as there is no file format for out of order messages; you must **Expunge**, even if there are no deleted messages.

You can also have Lafite automatically sort new mail:

`LAFITE.SORT.NEW.MAIL` [Variable]

If the value is `T`, Lafite always sorts new mail; if `:MULTIPLE`, Lafite only sorts when the mail was retrieved from more than one server. Initial value is `NIL`.

Note that even when using a single mail server, mail can easily be out of order, due to the varying speeds at which messages pass through gateways, etc. Sorting assures that the messages will appear in the order in which they were posted. Sorting new mail seems fast enough that it is probably always worth doing; hence, the `:MULTIPLE` option may be dropped at some future time (send in your votes now).

Of course, to do the sorting, Lafite must now actually parse the Date field of messages. This also allows Lafite to display the date in the browser in a canonical form, rather than one that varies with the whim of the sender's software. Lisp's date parser (IDATE) was substantially beefed up to handle a wider variety of dates. It now claims to handle all dates legal in RFC822 syntax (except the silly single-digit military time zones), plus a fair variety of other formats found in mailers of recent years. The parser changes are in a module DATEPATCH loadable separately from Lafite.

Lafite considers dates older than 1970 spurious, so as to avoid being fooled by messages from machines that neglected to set the time.

With a new representation of dates, Lafite also has a new table of contents format. If you browse a folder with an old-style toc, Lafite will print "(older format)" when it reads it. Next time you do an Update, the toc will be written in the new format. Old versions of Lafite cannot read the toc files saved by this new version of Lafite, so if you try to browse a new folder with an old Lafite it will be forced to parse the file from scratch.

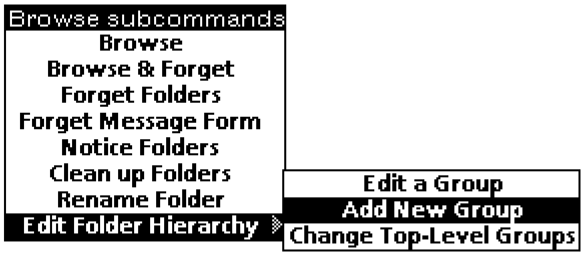
Lafite does not automatically parse dates on old messages, so until you issue a Sort command, old message dates will still be displayed as whatever string the old toc saved, not the new canonical form. In addition, when you *do* issue the Sort command, it must first go back and retrieve afresh all the date fields and parse them, an operation whose speed is dominated by the access time to your file system, much as parsing a folder from scratch.

Folder Hierarchy

This feature is based on a package written by Mike Dixon.

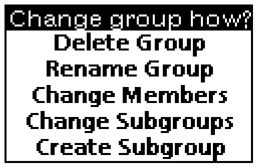
You can now organize your folders into a hierarchy, so that the folder menus presented by **Browse** and **MoveTo** are easier to use. Folders are organized into *groups*. A folder can be in any number of groups. Groups can have subgroups. The top level of the folder menu is composed of some or all of the groups, plus all folders that are not in any group.

The **Edit Folder Hierarchy** command on middle-button Browse lets you define new groups and change existing ones.



Edit Folder Hierarchy

Edit a Group The top-level command is the same as the subcommand **Edit a Group**. Lafite brings up a menu of all your defined groups. After you choose one, it offers you a choice of things to do to the group:



Delete Group does just that. Any folder in that group that is in no other group will reappear in the top-level folder menu. **Rename Group** lets you change the name of the group. **Change Members** prompts you with a menu of all folders, with the current group members preselected; you can add or remove folders. The folders in this menu are arranged with the ones not yet in any group clustered together to make it easy for you when you are first defining groups. **Create Subgroup** prompts you for the name of a new group, then which folders to put in it. The new group will appear as a member of the submenu presented by the supergroup, and its members in turn appear as a submenu to that. **Change Subgroups** lets you add or remove subgroups from the current group.

Add New Group Like **Create Subgroup**, but the new group appears at top level in the menu.

Change Top-Level Groups Prompts you to select which groups should appear in the top level of the folder menu. By default, groups created with **Add New Group** appear in the top level, while groups created by **Create Subgroup** do not. Regardless of your choice here, any group that is not currently a subgroup of another is included in the top level menu so that you don't lose it.

Note: the folder hierarchy is saved on your Lafite.info file, along with folder names and form names. Older versions of Lafite do not know about this structure, but those since June 1988 will at least preserve it. However, if you delete folders in an old version and then return to a new version of Lafite, the deleted folders may still appear in the hierarchy; you must manually delete them either by editing the variable `LAFITE.FOLDER.STRUCTURE` or by editing the file Lafite.info.

Mail Sending

There are two additional commands on the message composition window:

Reply To This command inserts a "Reply-to: *Your Name*" field in the header of the message. The name is selected for pending delete, so if you want replies to be sent to a different address, you can just start typing it. Lafite in Grapevine mode automatically prompts you to insert a Reply-to field at delivery time if one of the addressees has the syntax of a distribution list name (contains the character "^"). However, it can't figure this out for XNS or Arpa distribution lists, so you are strongly encouraged to insert the Reply-to field on your own with this command.

Change Mode This lets you change the mode of a message from GV to NS or vice versa. This can be useful if you clicked **SendMail** while in the wrong mode, or you want to Answer a message but do so from the other side of the fence. This command does not make any attempt to fix addresses already in the header to their other form; you have to do this yourself. The one exception is that if the cc or Reply-to field consists exactly of your mail name in the old mode, it substitutes your name in the new mode.

In addition, the old **Save Form** command has been renamed simply **Save**. This is in recognition of the fact that people seem much more often to want to save (checkpoint) messages in progress, than to create private mail forms. The message window now remains open after using the **Save** command (close it as you wish). When a previously saved message is ultimately delivered, you are asked whether you want to keep the saved form or delete it. There is also a menu item **Delete Message Form** underneath middle-button **Browse** for deleting a saved message explicitly. You can think of **Save** as a fancy version of TEdit's Put, which stores the message in your mail directory and remembers it (in the Saved Form menu) for a later Get.

You can now control whether Lafite bothers to ask you about whether to send a message formatted or not:

`LAFITE.SEND.FORMATTED` [Variable]

This variable is consulted when you ask to deliver a message that has what TEdit considers non-trivial formatting—font shifts, paragraph formatting, image objects, or NS characters. The choices are:

`NIL`	Send it unformatted.
`:ASK`	Prompt with a menu.
`T`	Send it formatted without asking.

The value can also be a list of elements (*formattingType answer*), specifying that when `(TEDIT.FORMATTEDFILEP` *message*`)` is *formattingType* the decision is *answer*. The initial value is

```
((NSCHARS :ASK)
 (CHARLOOKS :ASK)
```

```
                    (PARALOOKS :ASK)
                    (IMAGEOBJ T))
```

meaning to send messages with image objects formatted without asking (sending them unformatted is always a bad idea), but ask for all other kinds of formatted messages.

You can now send unformatted messages containing NS characters. However, note that versions of TEdit before Medley did not fully understand NS characters in unformatted text, so you should not do this if you expect that some of your Lisp recipients have not yet upgraded to Medley. If you never send messages to Koto or Lyric Lisp users (or don't care about this shortcoming), you might want to set the NSCHARS component of LAFITE.SEND.FORMATTED to NIL, i.e., have Lafite never send a message formatted merely because it contains NS characters.

Programs that compose message forms, such as those invoked from LAFITESPECIALFORMS, can specify that the message is to be delivered formatted without asking by giving the message (a textstream) the TEXTPROP property LAFITEFORMAT, value TEDIT. If the message should turn out not to be formatted after all, it is sent as plain text, just as if the LAFITEFORMAT property were not present.

You can also select the mode of the message by giving the message the TEXTPROP property LAFITEMODE. The value should be the mode, e.g., GV or NS. With no such property, the message is sent in the current mode.

Here are some more message customization variables:

LAFITE.SIGNATURE [Variable]

A string with which to "sign" your messages. This string is appended to the standard message forms—those produced by the **SendMail**, **Answer**, and **Forward** commands—always starting it on a new line. There should be a carriage return at the beginning of the string if you want a blank line to set off the signature. The initial value is NIL, meaning no signature.

LAFITE.GV.FROM.FIELD [Variable]

A string to use as the contents of the "From:" field of any Grapevine message you send that does not already have a From field. It must be a valid Grapevine address specification that parses down to your Grapevine identity, including registry. The usual syntax is "Human name <MailName.reg>", e.g., "Johann Amadeus Slonimsky III <Slo.pa>".

Note that unlike LAFITE.SIGNATURE, which appears in the message form itself, there is no visible evidence of LAFITE.GV.FROM.FIELD during message composition. Lafite does not use its value until it is time to add the Date and From fields when it delivers the message.

These two variables are examples of Lafite "personal" variables, in that they are closely bound to the logged-in user. The other is LAFITEDEFAULTHOST&DIR. If you change the logged in user (e.g., by calling (LOGIN)), Lafite resets all these variables to NIL, but remembers their values in case you log in again as yourself. This is designed to reduce confusion if you have more than one person using Lafite on the same machine. The values for each user are remembered as the value of the variable LAFITE.USER.INFO, an association list of user name and values, the latter being in property list format. Ordinarily you need not be aware of this variable, but you might, for example, want to initialize it with information about many users of a public machine.

Private DLs

Lafite now supports private distribution lists in outgoing messages. A private distribution list (dl) is an indirection mechanism: it allows you to name a possibly large group of addressees without explicitly enumerating them in the header of the message. Unlike a public dl, whose membership is maintained in the mail system's distributed database, a private dl is simply a file in which you have stored the real names. The names in the file must be in standard syntax for the protocol you are using (GV or NS), just as you would type in the header of a message, except that names may be separated with one or more carriage returns, rather than commas, to enhance readability. In addition, the Grapevine dl handler insists that the names be fully qualified, i.e., the Grapevine registry must be included. This is to eliminate confusion in the case where the current sender is in a different registry than the maintainer of the dl. Such a requirement would make NS dl's horrendously verbose, however, so we arbitrarily waive the restriction for them. Thus, be careful if you compose public NS dls.

The names in a Grapevine dl can make use of all the standard Grapevine syntax, which allows you to include comments in the file. For example, you could have a line reading

    <Carstairs.wbst>   "Works with bj on the ACME project"

Since NS header syntax does not support such elaborate syntax, comments are not permitted in NS dl files.

The file may be a TEdit file; Lafite strips out the formatting before attempting to parse the addresses in it.

The syntax for a private dl in a message is

To: *dlfilename*:;

i.e., the name of the file containing the real names, followed immediately by the two characters colon, semi-colon. In an NS message, this syntax is actually parsed as the fully qualified Clearinghouse name *dlfilename*:;:*defaultOrg*, and will appear that way to the recipient, but Lafite treats this kind of name specially. If the file name contains any characters whose syntax is significant—colon, semi-colon, parentheses, and brackets for Grapevine, colon for NS—the file name must be enclosed in double quotes. For example,

To: "{FS9:PARC:Xerox}<Carstairs>Mail>Friends":;

The file name need not be complete; Lafite fills in defaults to locate the file according to the following two variables:

`LAFITEDL.EXT` [Variable]

The default extension of private distribution list files. If a private dl is specified without an explicit extension, Lafite uses this extension when looking for the file. The value is initially "DL".

`LAFITEDLDIRECTORIES` [Variable]

List of directories on which private distribution list files may be found. When you use a private dl in a message, Lafite searches LAFITEDEFAULTHOST&DIR, then this list, for the file containing the addressees. The value is initially NIL.

Grapevine

Lafite now does a more thorough job of parsing Grapevine messages according to the RFC 822 specification and objecting to malformed addresses. It also preserves full addresses when composing replies, rather than discarding all the commentary.

`*GV-SHOW-POSTMARK*` [Variable]

When Lafite retrieves Grapevine mail, there is additional information about the transport of each message that Lafite normally discards as uninteresting, as it is for the majority of messages. When the variable `*GV-SHOW-POSTMARK*` is true, Lafite preserves this information in an extra line in the header of the message. The line reports the authenticated name of the sender, the time at which the message entered the Grapevine system, and the host from which it entered; for example,

```
GV-Info: Carstairs.pa at  4-Jun-88 23:38:00 from 204#16#
```

Note that the variable only affects the retrieval of mail, not its display. If the variable is `NIL` at the time of retrieval, the information is lost forever. A compromise is to set the variable true, but add "GV-Info" to the appropriate filter lists (`LAFITE.DONT.DISPLAY.HEADERS`, etc.), so that you see it only when you Unhide the header.

NS Mail

The NS mail retrieval code correctly interprets messages that are really returned messages. It now turns such messages into a description of a failed delivery, with a mail server as the apparent sender, instead of just putting a cryptic "TransportProblem" header in the message and leaving the header looking as though the message was from you.

The NS mail watcher no longer maintains a "session" on the mail server. This reduces server load a bit, and means that there is no confusion if another client (either Lafite or Viewpoint) tries to retrieve the mail of a user who happens to be running Lafite on another machine at the same time.

Formatted mail in NS mode is now retrieved more efficiently, by adjusting the formatting information directly, rather than performing higher-level TEdit operations that require doing a Get and Put. In addition to being faster, this avoids problems with messages that cannot be Put properly, such as those containing image objects unknown on the receiving side.

NS mail header fields are displayed in the order specified by the variable `NSMAIL.HEADER.ORDER`, independent of the order in which the mail system may have delivered them (the new Services 11 servers deliver some headers in an odd order).

Lafite now accepts messages sent in serialized format 3, which it turns out is identical to format 2.

NS Mail now handles "attachments" better. This change was actually made in a special release of NS mail in September, 1987, but is included here for completeness, and to explain the extensions that handle reference objects. Attachments are no longer left in your mailbox to be read later with, for example, Viewpoint. Instead, Lafite retrieves the entire attachment and encapsulates it into an image object that is enclosed as part of the text message, immediately following the header. A typical attachment appears in a mail message as:

Attachment: Viewpoint Document

If you click inside the object with any mouse button, you are offered a menu of things you can do with the attachment. The choices vary according to the type of attachment:

View as text This brings up a window in which is displayed the raw content of the attachment as ascii bytes. Runs of non-ascii bytes are replaced by nulls to reduce the amount of garbage. Some attachments are utter gibberish, but some, such as Viewpoint documents and Interpress masters, contain sections that are plain text. With this command, you may be able to decide whether you care to do anything further with the attachment. (Sorry, there is no Viewpoint to TEdit converter, nor are there plans for one.)

Put to file This prompts you for a file name, and creates a file to contain the attachment. The file must be on an NS file server for this command to be very useful; otherwise, information will be lost. Once the file is so stored, you can retrieve it from Viewpoint and manipulate it just as if you had originally retrieved it as mail in Viewpoint. If you are running in Lyric, you must have the module NSRANDOM loaded for this command to work.

Send to Printer This command is only available for attachments that are in the form of an Interpress master. The command prompts you for a printer (which must be one that accepts Interpress, of course), and sends the attachment to it for printing.

Expand folder This command is only available for attachments that are in the form of a "folder". A folder is a mechanism for collecting several objects into a single one. The **Expand folder** command splits the attachment up into its component objects, each of which can be manipulated in the same way as a top-level attachment. For example, if the folder contains an Interpress master, you can print it.

If you use the **Put to file** command on a folder, the name component of the file name you type will be treated as the name of a new subdirectory, and the components of the folder will appear as files in that subdirectory. For other types of attachments, **Put to file** (usually) produces an ordinary (non-directory) file.

If the attachment is a Viewpoint "Reference" object, the **View as text** and **Send to Printer** commands apply to the file the reference mentions, not the attachment itself. The **Put to file** command is renamed **Store reference**, to make it clear it applies to the reference object itself, rather than the file being referred to (which there is no point in copying, since Viewpoint can deal with the reference object itself just fine). References to folders also offer the command **FileBrowse**, which brings up a FileBrowser (if you have it loaded) browsing the directory referred to.

Messages containing attachments are otherwise just like formatted messages—you can move them toother folders, and you can forward them (assuming the mail is received by another Lafite recipient and did not have to pass through an information-losing mail gateway).

Lafite now sends plain text messages as "text attachments" rather than "mail notes", unless `*NSMAIL-SEND-MAIL-NOTES*` is true. This avoids a nasty bug in Services 11.3 mail servers and should be completely transparent to users.

There is currently only a crude mechansm for creating your own attachments to send to other users, as follows: The attachment must be a file on an NS file server. To send an attachment, place a line "Attached-File: *XNS Filename*" in the header of your message. To send the file as a reference (this sends only a pointer to the file, not the file's content, so the recipient must have access to your server), use the line "Attached-Reference: *XNS Filename*". You can have only one such line in the header, and the body of the message must be small enough to send as a mail note (less than 8000 characters). Probably the only interesting kind of attachment to send currently is an Interpress master, which both Lafite and Viewpoint recipients will be able to print. To "forward" an attachment you received in NS mail, you can use **Put to File** on the attachment, then compose a message with "Attached-Reference" or "Attached-File" referring to the resulting file. Incompleteness: Most of the attributes of the file are lost when you send the file itself as an attachment, rather than a reference. You cannot send directories as attachments.

Scavenger

Lafite's mail scavenger has been completely rewritten. It now handles most simple cases of bad message lengths by altering the file in place, rather than laboriously copying the entire file to a scratch location. The diagnostic output is more informative. The scavenger is also integrated with the browser, so that if you browse a malformed folder and the browser reports "unable to parse", you are immediately offered the option of scavenging.

You can still scavenge manually by calling `LAFITE.SCAVENGE`, but ordinarily you should never need to do this.

(`LAFITE.SCAVENGE` *FOLDERNAME ERRORMSGSTREAM FORGET?*) [Function]

FOLDERNAME is the name of the folder to scavenge, either a full file name, or abbreviated as it might appear in the folder menu. *ERRORMSGSTREAM* is an output stream to which to direct diagnostic information; it defaults to the terminal. If *FORGET?* is true, the folder is not added to the folder menu, just as if you had used the **Browse & Forget** command.

The old scavenger entry point `MAILSCAVENGE.IN.PLACE` is obsolete and has been removed.

Miscellaneous

The middle-button **Browse** command has a new subcommand **Browse & Forget**. This command prompts for a file name and browses it with the `:FORGET` option, i.e., it does not add it to the folder menu. Note that this command does not remove an existing folder from the menu, it merely refrains from adding a new one. Also, since it is not in the menu, it will be inconvenient to move mail into it—you have to select "Other Folder" and type the name each time. The command is intended for those times you want to browse some other user's mail folder that you do not plan to look at again.

Browse has another new subcommand **Rename Folder**, which you can use to change the name of a folder. This renames the file(s) supporting the folder and fixes various menus.

Lafite's table of contents code now handles messages containing NS strings in the Subject field. This fixes the bug where browsing such a file would claim that the toc was inconsistent with the folder, throw away the toc file and reparse from scratch. Also, the bug where Lafite reported "Folder is Empty" when discarding a malformed toc has been fixed.

The Lafite message file format has been very slightly extended to permit you to receive messages of up to 99,999,999 bytes (about 100MB!) in length. This means it no longer is forced to split apart messages longer than 99,999 bytes, which did very bad things to large Viewpoint attachments and sometimes caused loss of mail server connection. It also means that Lafite files are no longer exactly compatible with Laurel or Hardy format, in case anyone cares. As mitigation, however, messages that fit in the old format (99,999 bytes or less) are converted back to the old format whenever they are moved (either by **MoveTo** or **Expunge**), so you can arrange for a Hardy-compatible file by moving all messages into another folder (assuming your messages are all short enough).

Lafite now treats folder names as strings, and does its best to (a) ignore alphabetic case within folder names and (b) preserve the case that you type when creating a folder. This means, among other things, that you can now have mixed case in your `LAFITEDEFAULTHOST&DIR`, and that you can have folder names that are entirely numeric, such as "1186".

Lafite also keeps in its folder menu what the server considers the "canonical" name of each file to be. This means that if your folders include a file not on your default directory with a host not specified by its canonical name, then the next time you browse this folder, Lafite will think it is a new folder, leaving you with a menu containing both the old and the new names. Use the **Forget Folders** subcommand of **Browse** to dispose of such duplicates.

The information about your folders and message forms, formerly maintained in a file named LAFITE.PROFILE on your `LAFITEDEFAULTHOST&DIR`, is now kept in a file named LAFITE.INFO in a more extensible format. When you first run the new Lafite, if you have no LAFITE.INFO file, it will read LAFITE.PROFILE instead and convert to the new format.

Lafite does a better job now of noticing whether a mail file has been changed out from under you. It checks every time it opens the file, not just after logout. If the mail file has been changed but you have unsaved changes to the browser, Lafite will offer to save your changes if the mail file has merely been appended to. If, however, it has been expunged, so that the messages all live in different locations in the file, this is not possible, and the changes in your browser must be discarded.

When you return from logout and Lafite can't locate the file behind one or more of your browsers, it no longer closes the browser. The next time you try to do anything with the browser, such as display a message, Lafite will again try to locate the file. Thus, if the file server holding the file is down, you need only wait for it to come back up before using the browser, rather than having to rebrowse the folder.

The after-logout code now runs mostly in a separate process, which allows the call to LOGOUT to return faster.

Lafite files are now of type LAFITE, rather than TEXT, since they typically contain non-text content, such as NS characters and TEdit formatting. On any kind of server but an NS file server, type LAFITE is usually coerced to BINARY. You should be careful when copying old mail files (those that might have been written as type TEXT) to a server with a different end of line convention (e.g., between XNS and Unix).

LAFITE.HOST.ABBREVS [Variable]

A list specifying abbreviations for commonly-used host and directory combinations. This is useful if you regularly work with mail files on more than one directory. Each element of LAFITE.HOST.ABBREVS is a list (*abbreviation fullname*), where *abbreviation* is a string ending in colon (and not containing any other filename punctuation) specifying the abbreviation you want to use, and *fullname* is a string specifying the full host and directory name for which you wish the abbreviation to stand. You can use the abbreviation any place that Lafite prompts for a folder name, and Lafite will expand it accordingly. Conversely, Lafite will replace *fullname* with *abbreviation* when it puts folder names into menus, title bars and such. *abbreviation* can also be a list of abbreviations for the same directory, in which case Lafite will use the first element of the list when constructing abbreviated names, useful if you want the names to sort differently in the folders menu, or make the abbreviation stand apart from the rest of the name.

For example, if LAFITE.HOST.ABBREVS is

```
((("~Pooh~:" "Pooh:") "{pooh/n}<pooh>carstairs>mail>")
 ("Jo:" "{FS8:Parc:Xerox}<Josephine>Mail>"))
```

then if you type the name "Pooh:Active", Lafite will treat it as if you typed "{pooh/n}<pooh>carstairs>mail>Active", but will turn that full name into "~Pooh~:Active" when putting it in the folders menu. Lafite only substitutes for entire directories, so, for example, it would not abbreviate "{FS8:Parc:Xerox}<Josephine>Mail>Old>August" to "Jo:Old>August".

LAFITE.HOST.ABBREVS is one of the variables that Lafite caches, so it will not notice changes to it until you either restart Lafite or use the **Recache** command. And because host abbreviations take the syntactic form of a device field, you cannot use an abbreviation to stand for a directory that includes a device, e.g., {vax}dp1:<Fred>.

Changes to Programmer Interface

Well, actually, there is no documented programmer's interface. Numerous internals have changed; this section lists but a few.

With the new handling of modes, nearly everything surrounding the use of \LAFITEUSERDATA and the record LAFITEOPS (and hence the variable LAFITEMODELST) has changed. POLLNEWMAIL has been restructured; PRINTLAFITESTATUS behaves differently. There is a new hook LAFITENEWMAILFN. When this variable is non-NIL, its value is called, with no arguments, when the Lafite status window notes new mail.

The MAILFOLDER record has changed slightly. The use of its FOLDERDISPLAYWINDOWS field is different. The new field FOLDERDISPLAYREGIONS replaces BROWSERSELECTIONREGION.

The whole family of functions around \LAFITE.BROWSE has substantially changed.

Several functions have been deleted or renamed including LA.REMOVEDUPLICATES, LA.SETDIFFERENCE, PROFILEFILENAME, \LAFITE.MERGE.PROFILES (now \LAFITE.MERGE.NAMELISTS), CHANGEFLAGINFOLDER, \LAFITE.GETMAILFOLDER.

MAKEXXXSUPPORTFORM, when given an address in a-list form, chooses the first address in the list that is for a supported mode. Thus, you can arrange to have Lisp Reports always go to an NS address (assuming NS mail is loaded), by making (NS . "*NS Support Address*") be the first element of LISPSUPPORT. Special forms that call MAKEXXXSUPPORTFORM can also encourage a particular mode this way without having to switch the mode themselves.

Partial List of Fixed or Obsolete ARs

267	Easy font changing
307	Filter selected fields out of displayed msg?
552	Make it easier to change logged in user
1239	Recover from "Lafite is confused..."
1249	Want automatic GetMail on icon expansion
1273	Prompt for GV login if BadPassword error => Declined: use Login
1389	Answer command should preserve entire address
1644	Private dl's
2143	Want detachable display windows that survive Update
2421	Preserve case when creating files
2433	Use `GETBOXREGION` to place window?
2597	Check that GV headers conform to RFC822
3168	Want user-specifiable browser / menu layout
3174	List of regions for browser, display, etc
3269	Update, GetMail etc should check creationdate / eof and rebrowse if needed
4420	`LAFITEHARDCOPY.MIN.TOC` doesn't work with `LAFITEHARDCOPYBATCHFLG`
4628	addresses lacking a closing double quote break with illegal arg 65536
5109	Don't loop if bad NS password
5110	Blank name (e.g. in Reply-to) completes to *:domain:org
5111	Want name parser that doesn't add default registries
5116	Parser loops forever if header address has unmatched paren or quote
5117	Want way of dealing with NS Mail attachments without running Star
5121	Close / Shrink menu includes "Expunge" even if no deleted messages
5122	Simultaneously watch for both NS and GV mail
5125	Case sensitivity in `LAFITEDEFAULTHOST&DIR` screws up MoveTo
5129	Want browser title rearranged to avoid clipping important info
5134	Message from self not noted if user not authenticated when folder browsed
5137	Want to be able to keep display window open after browser shrunk
5300	Answer NS message in GV mode does not give To field
5541	Default status window off screen on small 1186 screen
5638	Prune headers on forwarded messages
6372	`LAFITENEWMAILTUNE` not on daybreak
6260	Blank To / cc line parsed incorrectly
6547	Can't put Arpa address in From field
6918	Send unformatted dies in `COPYBYTES` illegal arg
7244	Numeric folder names die in string-equal
8643	`(il:lafite 'il:off)` breaks in `SHADEITEM`, menu = `nil`
8679	Changing modes with Lafite off doesn't reset `\LAFITEUSERDATA`?
8995	When toc discarded, says "Folder empty"

There is a new version of Lafite. For Medley/LispCore users, it is on {Eris}<LispCore>Internal>Library>. There is also a version compiled for Lyric, but substantially less tested, on {Phylum}<LispLibrary>Lyric>New>.

For many users, the highlight of the new version is that GV and NS mode run simultaneously. Details of this and some of the other features and bug fixes may be found in the 15-page file Lafite-Jun-88.IP (same directories).

If you want to load the new version on top of the old, first Quit out of the old Lafite, then load *all* of the following `LCOM`s from the appropriate directory:

```
LAFITEBROWSE
LAFITEMAIL
LAFITESEND
MAILCLIENT
NSMAIL
LAFITEFIND
LAFITE
```

Of the three files `MAILCLIENT`, `NSMAIL`, `LAFITEFIND`, you need only load those that are already present in your sysout from the old Lafite; I believe all are present in the standard Parc sysout, so all must be reloaded. Note that `LAFITE.LCOM` must be loaded last. Lyric users: you must have `NSRANDOM` loaded (it is in the Parc sysout) for full NS Mail functionality.

Rooms users should additionally load {Pogo:AISNorth}<Rooms>Medley>Users>NEW-LAFITE-WINDOW-TYPES.DFASL and re-dump their suite. Henceforth NEW-LAFITE-WINDOW-TYPES will be autoloaded when their suite is loaded, provided {Pogo:AISNorth}<Rooms>Medley>Users> is on DIRECTORIES. There apparently isn't a version compiled for Lyric just yet.

This version of Lafite is incompatible with Lens; Ramana will send a message when there is news on that front. It is also unlikely to be compatible with any other program that thinks it knows about Lafite internals, as many have changed.

If the brave pioneers report no outrageous bugs in the next week or two, these will get moved to the standard release places.

Lafite To Do

Last Edited Sep 4, 1987, vanMelle.pa. Last ar = 8995.

Browse/SubBrowse commands

Delete Folder Command ?
Want Rename Folder command [ar 5140]
Browse empty folder => divide by zero [ar 8166]

Browser

Parse dates to provide uniform date syntax in browser window [ar 2412]
Sort [ar 365]
Redistribute vs. Forward (implies smarter Answer) [ar 1693]
Deferred MoveTo for speed [ar 392]
Parse EOL = CRLF files correctly [ar 730]
Imageobj in message header -> "24-bit extended NS encoding not supported" [ar 4635]
Want icon to be labeled to be more overlappable [ar 1242]
Default the Subject field to first line of message [ar 1828]
Want messages directly addressed to recipient to be highlighted in browser [ar 2441]
Want sub-browsers to allow scanning selected subset of messages [ar 2401]
Want "Get Mail" near "Display" [ar 4263]

Browser Marks

Want to use DEL instead of ESC to abort mark setting [ar 478]
Want to abort mark entry by clicking [ar 1071]
Want multiple flags per message [ar 54]

Browser Display

Scroll by integral number of lines
Want Display after Delete to happen in more cases [ar 2405]
Browser Title include number of messages [ar 2415]
Toc "From:" not show <addrs> or defaultregistry
Updating of mark does not draw delete line thru mark [ar 1072]
Want to control when browser chooses to scroll up [ar 907]
Want "End of Messages" indicator in browser [ar 765]

Browser Searching

Search commands [ar 571]
Want selection ops using "In-reply-to" field to jump to other msgs [ar 481]

Message Display

Don’t tie up mouse if it will take a while [ar 1387]
Want to be able to abort Display when it’s taking too long [ar 2498]

Hardcopy

Hardcopy options (#copies, #sides, font, printer, etc) [ar 941, 952, 2403]
Want Hardcopy forms like Laurel [ar 2439]
Want to be able to cancel pending Hardcopy [ar 2495]
Want batch hardcopy to run in a separate process [ar 5145]
**Don’t hardcopy too much at once.

Update Command

Have `LOGOUT` perform Update [ar 2414]
Maybe maintain "First deleted message" and/or "First changed message" to speed expunge and update.
Accelerator for Expunge bypassing Changes/Expunge menu? [ar 761]
Intercept `FILE SYSTEM RESOURCES EXCEEDED` during expunge [ar 2337]
Don’t leave Lafite in bad state if Expunge aborted [ar 2346]
Want faster update [ar 2427]
**Don’t Update shouldn’t try to close other files (server down) [14 jan 87]
**When shrink/close with Expunge, etc, shrink immediately, gray the icon. When finished, ungray/close the icon.

Retrieving Mail

Try harder to open connection to distant site [ar 2465]

GetMail breaks with Bad.State.For.Bin when mail has gotten archived [ar 5142]
Break on messages longer than 99999 bytes [ar 2345]
Choke on very long (9 piece) messages with bad.state.for.bin [ar 4604]
Programmatic GetMail [ar 2416]
Intercept errors during GetMail (disk full, bad.state.for.bout) [ar 2418, 2348]
Programmatic filtering of incoming mail [ar 1385]
Want to make it harder to GetMail to wrong folder [ar 3164]
Should inhibit Delete et al during GetMail [ar 2445]
9305 crash after getmail clicked during shrink/expunge [ar 4027]

Grapevine

Suppress/interpret "Server full" abort messages [ar 429]
Say "server full" instead of AllDown when appropriate [ar 479]
Appropriate errorset protection against grapevine; bad.state.for.bout [ar 173]
Notice messages examined by Ernestine? [ar 187]
Notice if messages deleted (discrepancy with what is printed) [ar 1198]
BAD.STATE.FOR.BOUT under GV.ADDTOITEM while sending message [ar 2517]
Break when retrieving archived mail fails [ar 5795]

NS Mail

Don’t break if can’t find Clearinghouse [ar 3049]
CLOSEF of closed stream when NS mail server not responding [ar 3160]
Delivery failed--(SERVICE.ERROR MediumFull) [ar 3223]
Delivery dies if mode switched at same time [ar 5131]
Space in From field not allowed in Arpanet mail [ar 5139]
**Break in ns.openmailbox the first time it is used--(difference n nil) [15apr87]

Message Sending

Require templates be filled in? [ar 2420]
Want alternative to Lafite Outbox for getting at old messages [ar 3169]
Better programmatic interface?
Want settable cc: self [ar 951]
Want cc to file instead of self [ar 1240]
Want blind cc [ar 5133]
Want nicknames [ar 1251]
Want faster Abort [ar 1733]
Unexplained failures—finishes instantly, or "something went wrong" [ar 1946, 2053]
If break under COERCETEXTOBJ, Date/From not discarded [ar 2706]
Don’t detach all windows—let TEdit do it.
Want composition window to shrink during delivery [ar 5138]
Don’t top delivery window so much [ar 7944]
Outbox confused: new msg in first slot, others cleared [ar 8075]
**\SENDMESSAGE.RESTARTABLE takes CAR of string [Pavel 6jul86]
**Reshaping delivery window during delivery gets ARG NOT MENU [24 Jul 86]
**Abort button has a scroll bar! [27 May 86]
**When delivering, shrink delivery window

Answer

Answer: want command to yank answered msg into buffer [ar 5123]
Answer command for multiple selection [ar 2406]

Quit/SubQuit

Want to be able to Quit Lafite without breaks in odd situations [ar 2521]

Mail File Maintenance

Partition mail folder menu by directory and subdirectory [ar 1130]
Require confirmation when forgetting folders, forms [ar 2480]
Don’t try to access {dsk} files on machine different from where they first lived [ar 2442]

Maintain

Want wildcarded facility for finding names in MAINTAIN [ar 3556]
Want Add Forwarding command [ar 5120]

Should make it harder to add invalid names to DLs [ar 5143]
Want command to locate all dl's containing user [ar 5149]

Mail Scavenge

MailFileScavenger [ar 623]
Offer to invoke scavenger on parse failure [ar 4213]
If parse fails, provide partial use?
Win if file ends in extra nulls
MP9318 when the final rename causes "File System Resources Exceeded" [ar 3836]

Documentation

LAFITEFIND

Other

Confusion from Idle logins [ar 4411]
Better recovery from file servers down [ar 4250]
Want status window not to resize [ar 5130]
Want MailFolder icon that stacks better [ar 2404]
Conflict between Lafite and BIG [ar 2718]
Shouldn't break under `\LAFITEDEFAULTHOST&DIR` when server down [ar 3166]
Want Lafite to close file if it's idle for long enough [ar 3162]
Want better error messages when profile vars (e.g. font) set badly [ar 2520]
Better recovery from lost file server after logout [ar 1388]
One-time font change for display or hardcopy
use `{TEMP}`
Edit Message [ar 2428]
Want MCI interface [ar 3104]
Command to put backtrace into message window [ar 614]
Protect initialization better from Hardreset [ar 2083]
Want programmers' interface to Mail and Lafite [ar 2462]
Add background menu command to turn Lafite on [ar 2497]
Closing a folder when file disappears should flush pending commands [ar 2456]
Browser got --- --- [0 chars], messages maybe lost [ar 3253]
Add font variables to `FONTVARS` [ar 6697]
Profile confusion when you move your default dir [ar 7244]
**Extend info in profiles [MikeDixon, 19Jan87]
**Reshaping new LafiteStatusWindow gets divide by zero in `RESHAPEALLWINDOWS` [10 Jun 86, 7jul86]
**Race: Move to new folder vs. Browse of that folder [1 Aug 86]
**Race: Move vs Update's choice of expunging or not [7 sep 86]
**Notice Files doesn't understand fs aliases [29 sep 86]
**Want number of messages on browser icon [12 Nov 86] ({qv}<briggs>lisp>lafitefoldericon)
**Cleanup Folders message not scrolled to be visible [3 Sep 87]
**`LA.LONGFILENAME` should handle pathnames [Rao, 19 Feb 88]

Not mine

Display should be faster [ar 60]
Viewpoint mail notes with NS characters read wrong [ar 4667]
Documentation: need load `MAILCLIENT` [ar 8282]

Browse/SubBrowse commands

File name prompt should allow spaces in name [ar 4270] 6/86
Don’t add new name to known folders if disconfirmed [ar 5113] 6/86
Want to enumerate all mail files in a directory [ar 5118] 6/86
Browse Laurel File broken [] 6/86

Browser Display

Want all browser prompt messages to go thru common routine [ar 3161] 6/86

Browser Searching

Find Previous All when current selection satisfies search says "found in 1 messages" if no other instances found [ar 2413] 6/86
"Find Related is unreliable" [ar 3190] 6/86
Want to be able to Find messages from self [ar 5115] 6/86
Search for Mark [] 6/86

Message Display

When "No more messages", at least topw the old msg [ar 5170] 6/86

Hardcopy

Error if close browser before hardcopy finishes [ar 2784] 6/86

Update Command

Don’t write scratch file on `{DSK}` if there isn’t space [ar 5124] 6/86
Want Expunge command not to move selection pointer [ar 5141] 6/86

Retrieving Mail

Graphical feedback during GetMail [ar 1466] 6/86

Grapevine

`SHOULDNT` in `MS.RETRIEVEOPERATION` [ar 2461] 6/86
Parse recipient names with respect to actual sender’s registry, not `DEFAULTREGISTRY` [ar 5144] 6/86

Message Sending

Separate formatting info [ar 426]
SendMail with Lafite off breaks under `ATTACHWINDOW` [ar 2985]
SendMail with Lafite off and mode not set fails [ar 3157] ☑
Error if have Image Obj in private form [ar 3163] ☑
`LAFITE.SENDMESSAGE` breaks attempting to print to nonexistent prompt window [ar 3165] ☑
Illegal arg if you send as plain text a message containing imageobj’s [ar 1244]
Document LafiteSupport et al variables, point elsewhere if appropriate [ar 2619] ☑
`NIL` displayed in outbox when no subject [ar 4180] 6/86
Better icon for unsent mail; include Subject [ar 5112] 6/86
Remove `BEFOREEXIT` prop if delivery aborted [ar 5146] 6/86
Formatted messages should not say "Format: TEdit" [ar 5171] 6/86

Browser

close, button "hardcopy only" doesn’t close [ar 1838] ☑
Changing mark character should imply Update needed [ar 2193] ☑

Quit/SubQuit

Quit should not flush unsent mail [ar 5135] 6/86

Other

`LAFITEMODE` dies with arg not menu `NOBIND` if Lafite not on [ar 2890, ☑ 3003] ☑
↑E out of NS Lafite password prompt gives 9305 [ar 3158] ☑
Status window too narrow for some "not responding" messages [ar 3159] ☑
Quit dies with udf `##CLOSE##` [ar 2831] ☑
Want protection against LAFITEMAILWATCH process being killed [ar 2339] ☑
Multiple LAFITEMAILWATCH processes remain when more than one Lafite user uses the same machine [ar 3310]
Want TEDITPROPS arg to \SENDMESSAGE [ar 2387] ☑
Saved forms should retain formatting info [ar 2407]
Save Form should not make duplicate names in menu [ar 2482]

"m" character marks should be moved [ar 2422]
Message with image objects should automatically be sent formatted [ar 2426]
Udf CLOSEMAILFOLDER under Browser Laurel File [ar 2429]
"unrecognized courier error message - 0" while sending NS mail [ar 2449]
Glitches in prompt for filename in Save Form [ar 2425]
Lafite ignores profile if file server unresponsive; later smashes it [ar 5119] 6/86
Default Status Window position inappropriate for 15" display [ar 5147] 6/86

Fixed in 6/88 Lafite

Browse/SubBrowse commands

Case sensitivity in LAFITEDEFAULTHOST&DIR screws up MoveTo [ar 5125]
When toc discarded, says "Folder empty" [ar 8995]

Browser

Want user-specifiable browser/menu layout [ar 3168]
Want browser title rearranged to avoid clipping important info [ar 5129]
Message from self not noted if user not authenticated when folder browsed [ar 5134]
**Want to be able to specify icon position.
**Want browser title to be short if possible. [19jan87]
**Want to be able to shift-select from browser [Shrager 4Mar87]

Browser Display

Want detachable display windows that survive Update [ar 2143]

Message Display

Easy font changing [ar 267]
Filter selected fields out of displayed msg? [ar 307]
Use GETBOXREGION to place window? [ar 2433]
Want to be able to keep display window open after browser shrunk [ar 5137]

Hardcopy

LAFITEHARDCOPY.MIN.TOC doesn’t work with LAFITEHARDCOPYBATCHFLG [ar 4420]

Update Command

Update, GetMail etc should check creationdate/eof and rebrowse if needed [ar 3269]
Close/Shrink menu includes "Expunge" even if no deleted messages [ar 5121]

Retrieving Mail

Want automatic GetMail on icon expansion [ar 1249]
LAFITENEWMAILTUNE not on daybreak [ar 6372]

Grapevine

Prompt for login if BadPassword error [ar 1273] => Declined: use Login
Answer NS message in GV mode does not give To field [ar 5300]
Check that headers conform to RFC822 [ar 2597]

NS Mail

Don’t loop if bad password [ar 5109]
Want way of dealing with NS Mail attachments without running Star [ar 5117]
Simultaneously watch for both NS and GV mail [ar 5122]
**Don’t hold onto session [19aug86]
Blank name (e.g. in Reply-to) completes to *:domain:org [ar 5110]
Blank To/cc line parsed incorrectly [ar 6260]

Message Sending

Cleanup mess of LAFITE[CURRENT]EDITORWINDOWS, MESSAGEFILE prop
Private dl’s [ar 1644] (see {QV}<Briggs>Lisp>LafitePrivateDl)
addresses lacking a closing double quote break with illegal arg 65536 [ar 4628]
Parser loops forever if header address has unmatched paren or quote [ar 5116]
Prune headers on forwarded messages [ar 5638]
Can’t put Arpa address in From field [ar 6547]
Send unformatted dies in COPYBYTES illegal arg [ar 6918]

Signature (For From, for end of msg).
**Want to answer the "formatted" question automatically [12 apr 86]

Answer

Answer command should preserve entire address [ar 1389]
Want name parser that doesn't add default registries [ar 5111]

Quit/SubQuit

Make it easier to change logged in user [ar 552]
`(il:lafite 'il:off)` breaks in `SHADEITEM`, menu = `nil` [ar 8643]

Mail File Maintenance

Preserve case when creating files [ar 2421]
Can't have numeric folder names

Other

Recover from "Lafite is confused..." [ar 1239]
List of regions for browser, display, etc [ar 3174]
Default status window off screen on small 1186 screen [ar 5541]
Numeric folder names die in string-equal [ar 7244]
Changing modes with Lafite off doesn't reset `\LAFITEUSERDATA`? [ar 8679]
Parse NS chars better.
**Proclaim profile variables special [Pavel, 24 jan 87]
**checklafitemailfolders gets non-numeric arg `NIL` in `IDIFFERENCE` [3apr87]
**Want programmatic interface to browsing, complete with window/icon positioning and shrinking [Gregor 29apr87]
**Prompt for icon position on mail folders
**After logout, don't close missing folders

UNIXMAIL

By: Bob Bane (Bane.mv@envos.Xerox.com)

INTRODUCTION

UNIXMAIL is a new mail sending and receiving mode for Lafite. It sends mail via Unix hosts using the SMTP mail transfer protocol and can receive mail either by reading a Unix mail spool file or by calling the Berkeley mail program.

INSTALLATION

Turn Lafite off, load the file UNIXMAIL, make sure UNIXMAIL is configured appropriately (check the settings of the variables below, and make sure any other modules UNIXMAIL may need are loaded), then restart Lafite. If you are running Lafite on a machine that is isolated from the Xerox mail environment, you will probably want to set the variable LAFITE.USE.ALL.MODES to NIL and call (LAFITEMODE 'UNIX) before you turn Lafite back on; this makes Lafite send and receive mail in Unix mode only.

CONFIGURING

See **SENDING MAIL** and **RECEIVING MAIL** below for the exact meanings of the variables you will be asked to set. **NOTE**: these variables are checked by Lafite at authentication time only. To change UNIXMAIL's behavior without stopping Lafite, change the variables and then make Lafite re-authenticate (middle-button on the **Quit** Lafite menu, then select the **Just re-authenticate** sub-menu entry under **Login**).

D-machines:

UNIXMAIL.SEND.MODE must be set to SOCKET and UNIXMAIL.SEND.HOST must be set to the name of a TCP host that will accept SMTP connections. UNIXMAIL.RECEIVE.MODE must be set to SPOOL and UNIXMAIL.SPOOL.FILE must be set to the pathname of your Unix mail spool file.

Unix-based emulators:

The default values of UNIXMAIL.SEND.MODE and UNIXMAIL.RECEIVE.MODE (PROCESS and SPOOL, respectively) will work if you normally send and receive mail from the machine where Medley is running. If your machine is on a non-trivial mail network, your machine may be sending and retrieving your mail from a remote machine without your knowledge; check all the variables below carefully if you have problems.

OTHER MODULES YOU MAY NEED

UNIXMAIL may need other library modules to work. The modules needed vary depending on what hardware you are using:

D-machines:

TCP is mandatory for Unix sending and may be used for Unix receiving, NFS is optional for Unix receiving

Unix-based emulators:

one of TCPOPS or UNIXCOMM is mandatory for sending

SENDING MAIL

UNIXMAIL can send mail in one of two ways, depending on the setting of UNIXMAIL.SEND.MODE:

UNIXMAIL.SEND.MODE [Variable]

If its value is the atom PROCESS, UNIXMAIL will send mail by doing SMTP with a Unix process-stream, normally running `/usr/etc/mconnect`. This option only works with Medley running on one of the Unix-based emulators.

If its value is the atom SOCKET, UNIXMAIL will send mail by doing SMTP with a TCP host. For this to work, an appropriate version of TCP must be loaded: either the TCP library module for D-machines or the TCPOPS library module for emulators that support it.

UNIXMAIL.SEND.MODE defaults to PROCESS.

Each of these send modes can be configured as well:

UNIXMAIL.SEND.PROCESS [Variable]

When UNIXMAIL.SEND.MODE is PROCESS, the value of this variable is the program run to create the SMTP process-stream. Initially the string `"/usr/etc/mconnect"` ; if your machine automatically forwards your mail to another host, you may have to put that host's name into the command, e.g. `"/usr/etc/mconnect fred"`

UNIXMAIL.SEND.HOST [Variable]

When UNIXMAIL.SEND.MODE is SOCKET, the value of this variable is the name of the host UNIXMAIL will attempt to contact via TCP to open an SMTP stream over socket 25. Initially NIL; on a Unix-based emulator this means to try the machine Medley is running on. This variable must be set when running on a D-machine.

RECEIVING MAIL

UNIXMAIL can receive mail in one of two ways, depending on the setting of UNIXMAIL.RECEIVE.MODE:

UNIXMAIL.RECEIVE.MODE [Variable]

If its value is the atom SPOOL, UNIXMAIL will receive mail by reading a Unix mail spool file.

If its value is the atom MAILER, UNIXMAIL will receive mail by running a Berkeley mailer as a Unix process-stream, normally `/usr/ucb/mail`. This option only works on Medley running one of the Unix-based emulators, and is a bit slower than SPOOL mode; it is primarily useful when you wish to occasionally switch between Lafite and the Berkeley mailer.

UNIXMAIL.RECEIVE.MODE defaults to SPOOL.

Each of these receive modes can be configured as well:

UNIXMAIL.RECEIVE.PROCESS [Variable]

When UNIXMAIL.RECEIVE.MODE is MAILER, the value of this variable is the program run to create the SMTP process-stream. Initially the string `"/usr/ucb/mail -N"`; the `-N` means to not print any banner or read any intialization file on starting the mailer.

UNIXMAIL.DONT.RECEIVE.STATUS [Variable]

When UNIXMAIL.RECEIVE.MODE is MAILER, the value of this variable is a set of message status letters; UNIXMAIL will leave behind any message whose status is included**.** Initially `""`, which means to read all messages regardless of status; another useful value would be `"O"` which means leave old messages behind.

UNIXMAIL.SPOOL.FILE [Variable]

When UNIXMAIL.RECEIVE.MODE is SPOOL, the value of this variable is the file UNIXMAIL will receive mail from. Any time this file exists, Lafite will say you have new mail; when Lafite gets mail from this file, it will read all messages in the file and then set its size to zero. Initially NIL; on a Unix-based emulator this means to try the file `"{UNIX}/usr/spool/mail/`*`username`*`"`, where *`username`* is the value of (UNIX-USERNAME). To access a Unix mail spool file from a D-machine, it will probably be necessary to load and configure either the TCP or NFS modules and then set UNIXMAIL.SPOOL.FILE appropriately.

Appendix A: Using Lafite Courteously

The great art of living easy and happy in society is to study proper behaviour, and even with our most intimate friends to observe politeness; otherwise we will insensibly treat each other with a degree of rudeness, and each will find himself despised in some measure by the other.

Boswell, *London Journal* (1762)

Foreward

This appendix is an edited version of ‘‘Message System Mores,’’ chapter 6 of the Xerox *Laurel Manual,* by Douglas K. Brotz, and the essay ‘‘Message System Mores’’ that Brotz published in *ACM Transactions in Office Information Systems*, Vol 1, No. 2. The material that appeared only in the ACM journal is copyright 1983 by the Association for Computing Machinery, Inc.

Douglas Brotz was a member of the team at the Palo Alto Research Center (PARC) that designed Laurel, which is an electronic message system similar to Lafite that was written for the Xerox Alto. Through his involvement with Laurel, Brotz discovered patterns of electronic message system behavior that may apply to Lafite users. He also developed some rules for appropriate message system behavior, i.e., message system etiquette. Because many Laurel and Lafite users have found this essay helpful, we have edited it for inclusion here, making the references appropriate for Lafite and deleting information that appears elsewhere in the Lafite manual.

Introduction

This is an essay on manners, in particular, message system manners. Electronic message systems provide a new mode of communication that, while offering convenience, speed, and reliable delivery, also opens channels that may be abused. At the Xerox Palo Alto Research Center, we have designed and implemented an electronic message system that has quickly spread throughout the Xerox Corporation. Through its use, we have discovered many patterns of message system user behavior that appear to apply to electronic message systems in general rather than to the particular system that we built. The focus of this essay is not on the features of our system, but on observations of user behavior in the electronic mail environment in an effort to spread understanding of this new medium and to instruct users in proper behavior.

The contents of this essay may be divided into roughly two kinds, objective observations of message system social phenomena and definitely biased suggestions of standards. The opinions expressed herein are solely those of the author. These opinions are not based on scientific studies or samples, but rather on certain intuitive feelings that have evolved through a close association with our system since its inception.

Communication Patterns

Part of the evolution of a society is the structure within which its members communicate. Face-to-face communication, both spoken and through gestures, has been with us for a very long time. Written communication and telephone communication have been employed for a substantially lesser amount of time. Nevertheless, these modes of communication have been around long enough to have developed certain standards of conduct and a framework in which reasonable communication can take place.

The electronic message medium has existed for a much shorter period of time, perhaps 20 or so years. (I am purposely ignoring telegraphic communication, which has very different characteristics due to its long delays and high cost.) Electronic message systems on personal computers have been available for even less time, probably less than 10 years. In this time, standards of electronic communication have not yet had time to mature, so we are still groping toward a workable electronic-messaging society.

In any of the mature communication media, each society places limits on what is considered acceptable behavior. Vulgar language or gestures are generally frowned upon in face-to-face communication, except in smaller sub-societies in which this mode of behavior is necessary for group membership. Shouting at close range is similarly considered to be in bad taste. Methods of dealing with such behavior in face-to-face communication run from mild rejection of the speaker to complete avoidance of that speaker in the future. As the number of human societies is large, and each has had much experience with this means of communication, the means employed for dealing with such situations are quite varied. Within each group, however, the methods used can be quite effective in stifling unwanted behaviors.

I will list several kinds of situations that arise in the electronic message medium and means for dealing with them. Where possible, I will draw parallels to other more traditional modes of communication to illustrate acceptable manners. In addition, I will try to point out the ways in which communicating via electronic mail is different from the traditional communication media, and how this modifies the problems to be dealt with.

The Wrong Number

We all have dialed wrong numbers and received calls from people who have dialed wrong numbers. The protocol for handling such situations is simple, and arises naturally as a result of the way in which standard phone calls are initiated. A typical wrong number dialog may be as follows:

Callee:	Hello.
Caller:	Hello. May I speak to John?

Callee: There is no one at this number by that name. I believe you have the wrong number.
Caller: Oh. Isn't this 555-1234?
Callee: No, it isn't. (And sometimes . . .) This is 555-4321!
Caller: Thank you. I'm sorry to have bothered you.

In postal communication, receiving misaddressed mail or mail for a former resident who has moved is akin to the telephone's wrong number. The post office's suggested remedy is for the recipient to line out the address and remail the letter. The post office will then attempt to forward the letter to the correct address, deliver it to the proper address, or return the letter to the sender.

Note that in both of these situations, it was not necessary to begin the actual conversation or open the letter. Enough information is exchanged at the outset to determine if the parties in the communication are the correct ones. This is usually not true when comunicating via electronic mail.

In electronic message systems, it is seldom the case that a message sent to a particular name is actually delivered to a recipient with a different name. A different situation is (unfortunately) common when a recipient has a popular name. The problem is that several people may have the same last name, and some electronic message systems do not have convenient facilities for mapping a person's actual name into that person's message system name. Thus, a person named Doe may receive mail for ADoe, BDoe, etc. Here, the original error is committed by the sender, who did not consider that ADoe's message system name was actually ADoe, but just assumed that it was Doe.

The parallel to this situation in the telephone medium is actually a bit more elaborate than the dialog given above. It is more like:

Callee: Hello.
Caller: Hello. Is Johnny there?
Callee: Hold on, I'll get him.
John: Hello?
Caller: Hey Johnny, let's boogie on down to the hoedown.
John: Who is this?
Caller: Come on buddih, this is good old Bodine!
John: I don't know any Bodine.
Caller: Oh. Ain't this 555-1234?

and so on. Notice that in this case a partial name match has occurred, and it is only later in the conversation that one of the parties discovers that something is awry. In the electronic mail case, it is nearly always the case that the message must be at least partially read to determine that it has reached an incorrect recipient.

This situation can be (and has been) handled in several inappropriate ways. First (and worst), the incorrect recipient can just ignore the message. No one gains through such inaction. Second, the incorrect recipient may send a response to the sender of the form ‘‘Stop sending me this trash!’’ This is a bit more helpful, but not quite the best that can be done. Third, the incorrect recipient may send the correct recipient a message of the form ‘‘Tell your senders what your name is!’’ This is not even as good as the previous response, as a message system user cannot know all possible senders.

Proper consideration by all involved can alleviate the ‘‘wrong number’’ syndrome considerably. Senders of messages should know their recipients. When sending a message, if you are not sure of a person's message system name, look it up. At Xerox PARC, the phone list has everyone's message system name correctly listed. Perhaps other organizations should do the same, and eventually a message-system-wide ‘‘white pages’’ may be published. Such lists help, but not if the senders don't use them.

A message addressed to an individual tends to be more important than a message addressed to a distribution list, in that a reply from an individual is expected more than replies from anonymous members of distribution lists. The names contained in distribution lists are usually correct, so there is generally no misdelivery problem. However, senders type the names of individual addressees for important messages directly. Thus, when there is a misaddressed message, it is generally an important one.

When you realize that a message is not for you, use the Forward command to send it back to the sender along with your polite comment that the message has reached a ‘‘wrong number.’’ Forwarding the message back is important, as the sender may not have a copy of that message any more. Once you have determined that you have received a ‘‘wrong number’’ message, *stop reading it.* A message sent through the message system may have personal material, and it is none of your business to peruse the entire message. (Many users who typically dispose of their received mail at a rapid clip take great delight in reading every last character of a misaddressed message—indeed, they consider it their solemn duty to do so.) It is for this reason that I do not suggest forwarding the message to the proper recipient. Determining who is the proper recipient is the job of the sender. It is presumptuous to believe that you know who the proper recipient is; you may actually forward the message to yet another incorrect recipient. Besides, determining the correct recipient may require reading more of the message than you ought to read. (If you think you know the message system name of the correct recipient by the time you realize that you are not the correct recipient, then you might include that name in your short covering note back to the sender. However, the mistaken sender should not expect correct identification of the intended recipient, just as he or she would not expect it in the telephone or postal mail systems.)

Some further points to consider are these. The ‘‘wrong number’’ mishaps generally happen to people who have common names and whose system names are exactly their last names. The honor of having one’s system name be exactly one’s last name is generally historical (‘‘I was the first Doe hired here, therefore I’m entitled to be Doe.PA forever!’’) A reasonable solution would be that our system administrators ensure that no user has a name that is a suffix of another’s, so that when ADoe arrives, then Doe has his or her message system name changed to BDoe, or whatever. In this way, the existing message system facilities will catch messages sent to Doe and return them as having been sent to a nonexistent name, at which point the sender can look up the correct message system name.

Rudeness and Vulgarity

The electronic mail medium joins several disparate properties of other communication media in an interesting way. The display of mail on a personal computer is a rather personal experience. Certain feelings of privacy and ownership pervade a personal computer user’s relationship with his or her machine. Thus, the process of reading one’s own electronic mail includes many of the personal aspects of face-to-face communication.

On the other hand, sending electronic mail is much more impersonal. The recipient is not present, and nearly none of the social strictures that govern one’s face-to-face communication are present. The sender is also able to speak his or her piece completely, without any intervening exchanges with the recipients that might moderate the entire business. This situation is enhanced when the recipients are not named directly, but are addressed indirectly through an impersonal distribution list. This imbalance in attitudes between sender and recipient has wide-ranging consequences.

One obvious consequence of this imbalance is that opinions expressed and the language used to express them in messages can be wildly inappropriate to the customs and expectations of the recipients of such a message. A reader may justifiably feel slapped in the face by a message he or she considers to be in extremely bad taste.

An interesting feature of the most annoying messages is that they tend to come from some ‘‘other’’ part of our messaging community. Julian Orr at PARC has observed that the most serious exchanges of antagonistic messages in our network have occurred shortly after some previously isolated message communities have joined. When the two societies meet and exchange messages, for some period the tone of the ‘‘other’’ community’s messages has offended members of ‘‘our’’ community. After an adjustment period, the two communities come to some understanding and establish norms for their intercommunication. This understanding typically involves identification of subjects whose discussion will cease to cross community boundaries.

The development of the junk mail lists in our Palo Alto and El Segundo, California registries illustrates this point. Both communities established Junk^ distribution lists for people interested in any for sale ads, announcements, random comments, etc. A slow link between these two locations was replaced by a faster one, and the volume of message traffic between the two communities increased dramatically. The two lists, Junk^.PA and Junk^.ES, were combined into an AllJunk^ list. However, even though the stated purpose of both Junk^ lists was ‘‘anything goes,’’ many PA registrants felt the ES junk mail was beyond the bounds of good taste. In other words, ‘‘their junk is much worse than our junk.’’ A majority of the original members of these lists withdrew from the Junk^ lists, and many splinter lists developed, ranging from Whimsy^.PA (‘‘lighthearted mail for PA’’) to @CrankMail.DL (a widely used private list in El Segundo). This is a history of two communities that seem to have similar characteristics. The implications for new message networks linking quite disparate communities are that more serious problems are likely to develop.

When rebuked for inappropriate behavior, errant senders have been known to say ‘‘I didn’t intend it that way!’’ This is not good enough. The damage has already been done. The only remedy is for senders to think about what they are saying and to whom they are saying it. The message system to date has been fairly unrestricted. Only as long as the society of message system users practices self-restraint will such a freewheeling communication medium be tolerated. There are several means of applying institutional censorship to the message system traffic, means that we hope will never need to be implemented.

Message System Costs

Many of the problems associated with improper use of the message system are exacerbated (caused?) by the lack of charging for message system usage. In nearly all other modes of communication, ‘‘sending a message’’ implies a certain cost (or risk), which rises with the number of recipients that are being reached. Free speech is, in this sense, not free at all. Certainly in a free society, one can say what one pleases, but not without paying for the means to say it. Let me illustrate this with some examples.

In nearly every communication medium, costs for the use of that medium are borne by the sender of messages. Postal mail requires the sender to pay for a stamp for each copy of a message that is sent. Telephone service is charged to the originator of calls, and each call (in general) goes to only one recipient. Broadcasting messages via radio or television requires a large investment on the part of the sender. The costs of printing handbills or posters are likewise borne by their authors. Public speeches, if they are to reach a large audience, require use of sound systems, etc., that are paid for by the speaker.

It may be argued that recipients do pay some of the costs for using some of these systems. However, these costs (the price of a radio receiver, basic telephone service, etc.) are generally constant; they do not increase as received message usage increases. A receiver's cost for electronic mail is similar in this respect in that the cost of a workstation on which the message system runs is borne by the receiver.

Some other modes of communication do require explicit payment by the receiver. Commercial films, books, magazines, and records fall into this category. However, publication of these materials does involve a substantial financial risk. Material that is not likely to be well received is seldom published, and when it is, large costs are often incurred by the publisher.

Electronic mail as it is usually implemented has a very different cost structure. The cost for a sender is minimal. It essentially consists of the time it takes to compose and send a message. If time is considered the major cost factor, then it is the recipients who pay dearly for the messages they receive. When the amount of time each recipient spends on a message sent to a large distribution list (even if a quick scan of part of the message followed by a Delete), is summed over all recipients, this is easily much more than the time consumed by the sender of that message.

While we would like to keep the free structure of a message system, where any user can send any message to any other users, this freedom must be used with some care. When electronic message systems become widespread, they will undoubtedly change their cost structures to match those of the more traditional communication systems.

Unsolicited Mail

The existence of large public distribution lists in our message system makes it easy for a sender to reach a very wide audience. Each distribution list has a distinct purpose, e.g., lists of people interested in particular topics, lists of employees in certain organizations, lists of members of particular projects, etc. Some lists are used primarily to keep track of all users of the message system. These include such lists as AllPA^.PA, AllES^.ES, etc., which contain the names of all individuals in those particular registries. There are also some lists maintained on a purely geographical basis, e.g., PaloAlto^.PA, which lists all message system users in Palo Alto, California. This is not necessarily the same as AllPA^.PA, which includes people in the PA registry, but who may not actually work in Palo Alto.

The audiences addressed by these lists should not be considered captive audiences for all users of the message system. At Xerox PARC, the purpose of any distribution list may be discovered by any user by running the Maintain program. Although all lists are (currently) available for use by any message system user, many lists, e.g., All*N*^.*N* where *N* is a registry name, should not be used by anyone who doesn't have a very good reason for doing so.

Many distribution lists exist for the enjoyment of their members who wish to receive items of interest to them. One should feel free to send an announcement of an upcoming musical event in Northern California, for instance, to Music^.PA. Such a message is quite inappropriate to send to AllPA^.PA, PaloAlto^.PA, etc. There are lists of message system users who have agreed to suffer through any and all messages. These lists (Junk^.PA, various @CrankMail.DL files, etc.) are the only lists to which ridiculous messages may be sent without incurring the justifiable wrath of message system users.

A message system user should understand when a message is appropriate to send to all people in his or her work group. Social values are different in different locations, and the members of each group should understand what they are. It has been observed that messages that are sent to audiences wider than the sender's immediate group are the ones that cause the most trouble.

Unfortunately, unsolicited messages have continued to be sent to inappropriate lists. Examples of inappropriate messages for standard organizational or geographic lists are:

''Does anyone know how to get my Alto fixed?''

''This is to let everyone in the message system world know that my phone number has changed.''

''I want everyone to know that I really like my roofing contractor.''

I'm sure that each user of the message system can recall some other similar gem. The following sections explore some of the consequences of unsolicited mail.

The Chain Reaction

To add insult to injury, after some piece of particularly ridiculous mail has been broadcast to an inappropriate audience, it invariably follows that some recipients cannot control their urge to make even bigger spectacles of themselves by sending their two cents' worth to everyone who received the original nonsense. While the original event is thought by many message system users to be annoying, the latter is considered to be downright stupid. Remember that once you push the Deliver button and watch the last chance to cancel fade away from your screen, there is no way to erase your comments from the collective memory of your peers.

I would like to list some of the typical responses that have been sent not just to the original perpetrator, but to the entire list of victims.

''Your message is inappropriate to send to all these good people.''

''If you don't like junk, then get off Junk^.''

''How do I get off Junk^?''

and, my favorite,

''Do you realize that if all of us replied to all of us (as I am doing right now) that the number of messages that would be sent would exceed the number of atoms in the known universe . . . .''

It is my opinion that bombarding only the original sender of a ridiculous message with equally nonsensical replies is poetic justice.

Use the Reply-To field to counteract the chain reaction phenomenon. However, there are situations in which replying to the entire list of original recipients is appropriate. In these cases, send the message without a Reply-To field, so that recipients who use Answer will get forms with all recipient names and lists included as recipients.

One final note on this topic. Although Lafite provides these mechanisms to help break chain reactions, the ultimate responsibility for messages sent lies with their senders. Always check the list of recipients in any message you are about to send.

Off-the-Record Responses

There are many situations in which a user submits a question to a wide audience, say to a distribution list of people interested in such questions, and indicates that he or she will collect responses and later make them public. This is a most reasonable thing to do, and it helps to reduce the chain reaction effect. Be sure to include a Reply-To: >>Self<< field when performing such services for your audience.

A note of caution is in order here. Messages should be considered private unless otherwise indicated. If your intention is to publish the responses, then by all means make that intention clear in the same message that poses the original question. If your message did not make that intention clear, and you decide that you would like to publish the responses, then follow up on each response asking whether you may do so.

If the intention to publish responses is clearly indicated in the original message, then publication of any response is fine, as long as that response does not explicitly mention that it should be considered private.

Masquerading

On occasion, people have received messages from fictitious senders, or even worse, from someone masquerading as another real message system user. This is a most serious breach of message system etiquette, and should be considered so by all message system users.

A fictitious From field is legitimate when a valid Sender field is included. For instance, messages that are properly signed with an organization’s name, say ‘‘The Lafite Group,’’ may be sent by explicitly typing a ‘‘From: The Lafite Group’’ line in the message header. Lafite will notice that a From field is already there, and it will include a Sender: *User Name* line in the delivered message instead of its usual From: *User Name* line. Any time you receive a message that has a strange From field, you may check the Sender field for the actual sender.

By a ‘‘masquerader’’ I mean someone who subverts the normal mechanisms embedded in the standard message system programs to send messages of dubious value, without having his or her name appear in such messages. This action is possible not only in electronic message systems, but in other more traditional communication media as well. Masquerading as another may be a criminal act when committed using traditional communication media, with penalties specified in laws that prohibit libel, slander, and fraud. Other situations, such as telephone ‘‘breathers,’’ are similarly outlawed.

Masquerading is the most serious social problem that we deal with in our message system. It occurs very rarely (three times to my knowledge), but when it does, our message system administrators make every effort to discover the perpetrator. The consequences may be serious, and the discovered perpetrators have all apologized publicly. I quote without reference from one such apology made by a masquerader who saw the error of his ways: ‘‘It is clear that any abuse of the message system, however lighthearted the intent, has repercussions far beyond what one might expect.’’

At this time, I do not know of any court cases involving libel, slander, etc., in an electronic mail context. Such cases are sure to arise when electronic mail does become more widespread. Masquerading in the message system is not cute or clever. Don’t do it.

Wizards Versus Naive Users

This section is addressed mainly to the wizards who should know better. The population of message system users covers a broad range, from those who have knowledge of the most arcane details of a system to those who just barely understand the basics of using that system. When you send a message to a wide audience, be considerate of the naive users, who may be confused by technical jargon.

This admonition extends to those who are using a new, restricted program. It does not help a recipient to hear ‘‘Oh you’re using that old program. Well, I guess you’re stuck.’’ Just don’t mention such things to users who cannot take advantage of them.

The Moral of This Tale

The moral of all this is simple: Be considerate. As we strive toward this goal, everyone’s use of the message system will become even more of a joy than it already is.

APPENDIX B: USING LAFITE AT XEROX

The main part of this manual tells you how to use Lafite at client companies. However, Lafite may be used somewhat differently at the Xerox site where you work. For example, you may use a message transport system known as Grapevine instead of the NS system; or you may use both systems. Or your site may use both Lafite and Laurel (a mail system similar to Lafite that was written for the Xerox Alto). This appendix gives you the specialized information you may need to use Lafite within Xerox.

Lafite use at Grapevine sites is described first. This section includes the documentation for Maintain, the program used to maintain Grapevine public distribution lists. It also tells you how to send Lafite messages to ARPANET recipients. Next described is Lafite use at sites having both the Grapevine and NS systems. Then there is a short description of some Lafite-related packages currently available only within Xerox. The final section describes the differences between Laurel and Lafite.

Note: this appendix was not indexed due to the difficulty of creating and maintaining separate customer and internal indexes.

The Grapevine Mail System

For about five years most Xerox sites have used the Grapevine electronic message transport system; it was for this system that Lafite was originally developed. At the present writing, Xerox is in the process of converting from Grapevine to the NS system. Some sites use only NS mail, some Grapevine, and some both. For the most part, you use Lafite with the Grapevine system in exactly the same way as with the NS system. The four major differences are the mail protocols, the forms of recipient addresses, the ability to send mail to ARPANET recipients, and the way in which public distribution lists are maintained.

Grapevine Protocols

The Grapevine system uses its own set of mail protocols, which you can obtain by loading the Library package file MAILCLIENT.DCOM. Simply type (LOAD '*{FILESERVER}<DIRECTORY>*MAILCLIENT.DCOM in the executive window.

Grapevine Recipient Addresses

A Grapevine recipient address has two parts separated by a period, in the form *Recipient.Registry*. The first part is the identifier for the recipient, usually that person's last name. If there are two or more people with the same last name in the same registry, each is usually identified by his or her first initial plus the last name.

The second part of a Grapevine address is the registry name. A registry is no more than a device for grouping related names. The registry name helps the mail system determine which mail server has the in-box for the recipient. Most registry names correspond roughly to ‘‘campuses’’ of activity within Xerox, which should make them easier to remember.

Lafite allows you to omit the registry name for recipients who are in your registry. You can find out what your local registry is by typing DEFAULTREGISTRY at the prompt in your executive window. DEFAULTREGISTRY is the global variable that names your local Grapevine registry; it is used if your log-in name does not include a registry. This variable is normally set in your initialization file.

Grapevine Registries

At present, the following registries exist:

ARPA	A pseudoregistry used to forward mail to people on the ARPANET and points beyond.
AUTO	A registry for software-driven functions requiring automatic access and Grapevine authentication.
CHOLLA	A registry for PARC/ICL, Palo Alto, California (no individuals in this registry).
DC	A registry for the District of Columbia.
DLOS	A registry for Dallas, Texas.
EOSA and EOS	A registry for associates or affiliates.
ES	A registry for El Segundo, California.
FX	A registry for Fuji Xerox, Japan.
GUEST	A registry for Palo Alto, California.
HENR	A registry for Henrietta, New York.
LB	A registry for Leesburg, Virginia.
OGC	A registry for the Office of the General Counsel.
OSDA and OSD	A registry for associates or affiliates.
PA	A registry for Palo Alto, California.
PASA	A registry for Pasadena, California.
RX	A registry for Rank Xerox, Great Britain.
SIEMENS	A registry for the Communications and Information Systems Group, Private and Special-Purpose Networks Division at Xerox XSG, OSD, Palo Alto, California.
STHQ	A registry for the Xerox Corporate Headquarters, Stamford, Connecticut.
SV	A registry for Sunnyvale, California.
WBST	A registry for Webster, New York.
X	A registry to be used by people in many geographic locations.
XRCC	A registry for the Xerox Research Centre of Canada, Toronto, Canada.

Grapevine Distribution Lists

The Grapevine system supports over 1,500 distribution lists, some of which are work related and some of which are purely for fun. You can obtain a complete distribution list directory, giving the name of each list plus a short description of its purpose, by printing {Indigo}<Registrar>GV>DLMap.Txt.

Grapevine distribution list recipient names end with the character ‘‘^,’’ in the form AISBU^.PA.

The public distribution lists for each registry are stored on mail servers and are maintained by the administrators of that registry, by the owners of the lists, and in some cases by the members of the lists. To create a public distribution list you must contact an administrator for your registry, who will make sure that your proposed name does not conflict with any others, and who will create the list for you. You can get your name added to appropriate lists by contacting the owners of those lists or by using the Maintain program (see below). If you cannot make a desired change to a public list yourself, send a message to the owners of the list. In Grapevine registries, by convention, the owners of a list named *List^.Registry* are listed in the companion list Owners-*List^.Registry.*

Delivery to Distribution Lists. If a recipient list includes public distribution lists, you must confirm the delivery by including a Reply-To field in the message (see chapter 9). If you do not include a Reply-To field before selecting Deliver, Lafite will give you a special Reply-To Field menu, asking you to choose between four options. The options are to send the message as is, have recipients reply to yourself, have them reply to someone else, or to abort the message. This is intended to minimize unintentional deliveries to large distribution lists.

ARPANET Messages

The ARPANET (Advanced Research Projects Agency Network) is a network linking ARPA research sites, including some Xerox sites. The Grapevine system is connected to the ARPANET, so you can correspond with individuals at ARPA sites outside Xerox.

As of this writing, ARPANET individual and distribution list addresses are in the form *Name@Host.*ARPA. However, all ARPANET addresses are gradually being changed to domain names. The primary Xerox address, ’’Xerox.ARPA,’’ will soon become ‘‘Xerox.COM’’; external ARPANET addresses are also being changed. The name changes are not expected to greatly inconvenience Xerox ARPANET users. The current Xerox address will be supported for some time after the change, so you can receive mail from people who don’t yet have your new address. If you send mail to an obsolete address, the ARPANET mail gateway will insert the proper address in the message header, but it is a good idea to keep an eye out for address changes in the correspondence you receive and update your private list of addresses accordingly. You don’t have to update ARPANET addresses in any Grapevine distribution lists you own, because ARPANET names on Grapevine distribution lists are kept current by Xerox network administrators (who periodically run a program to update all the names). If you want to verify an ARPANET address, you can search for it in the ARPANET host table stored on {Indigo}<NetInfo>ARPANET>Hosts.Txt. The first name in a host entry list is the primary name and the one you should be using.

ARPANET messages sent via Grapevine undergo some special processing. Because many mail systems receiving the mail you send over the ARPANET cannot process the long lines contained in Lafite messages, carriage returns are automatically inserted to make each line less than 72 characters long if there are any lines of more than 90 characters within the first 5,000 characters of the body of your message. If you have already inserted carriage returns to make your lines slightly longer than 90 characters and you don’t want your message to look unaesthetic, include a ‘‘Line-Fold: No’’ line in your header.

ARPANET Distribution Lists. ARPANET distribution lists are not maintained at any one, central location. Instead, they are administered locally by interested sites; for example, SF-Lovers@Rutgers is maintained at Rutgers. When you send a message to an ARPANET distribution list address, it goes to the appropriate host, who sends it to all the members of the list. Some hosts anthologize messages for certain lists (such as SF-Lovers); that is, someone at the host site periodically collects messages from members and sends anthologies of them out to everyone on the list.

Almost every ARPANET distribution list has a Xerox-internal redistribution list, so that all Xerox members are actually members of the redistribution list. For example, the redistribution list for SF-Lovers@Rutgers is XeroxSFLovers^.PA. You use Maintain to add yourself to the redistribution list of an ARPANET list in the same way you would add yourself to a Xerox distribution list. If you want to add yourself to (or make another administration request regarding) an ARPANET list that does not have a Xerox redistribution list, send a message to *DistributionList*-Request@*Host.*

Other Forms of Grapevine and ARPANET Addresses

Lafite recognizes three special forms of address that contain arbitrary text in addition to a valid Grapevine or ARPANET address. The first form is *Human Sensible Name* <*Actual Address*>. For example, if you wanted to include Cheryl Jones’s first name in her address you could write it as Cheryl Jones <Jones.PA>. The area outside the angle brackets can contain any text except for certain prohibited characters: parentheses, angle brackets, commas, colons, semicolons, and quotation marks.

The second form of address is *Actual Address* (*Comments*). You could, for instance, send an invitation to Jones.PA (and spouse). You can include any characters within the parentheses except additional parentheses.

The third form of address is ‘‘*Comments*’’ <*Actual Address*>. For example, you could send a message to ‘‘Special Attn: Sybalsky, Shih’’ TEditFriends^.PA. Any of the prohibited characters can be included within the quote marks except additional quote marks.

Maintain

Maintain is a Lisp Library package used only within Xerox for maintaining Grapevine public distribution lists and changing your Grapevine password. To load Maintain, type (LOAD ’*{FILESERVER}<DIRECTORY>*MAINTAIN.DCOM) in your executive window. To enter Maintain, type (MAINTAIN). Maintain will try to log you in automatically with your current user name and password. If that fails, then you may log in using the Login command in Maintain (see below).

How to Issue Maintain Commands

Maintain has its own prompt sign, in the form GV:. To issue a command in Maintain, you must type after the prompt sign only the initial letters that uniquely identify that command; i.e., as soon as enough of a word (usually one letter) has been typed, Maintain will complete that word for you. For instance, when typing the Add

Friend command, you need only type A F. Throughout this section, the letters that Maintain fills in are given in square brackets.

Once you have issued a command for a specific distribution list or individual, Maintain will fill in that same name for the next command you type. If the name is appropriate for that command, press the space bar or carriage return to confirm it. If the name is not appropriate for the command, begin typing the correct distribution list or individual's name to replace it, ending by hitting the space bar or carriage return.

The Login Command

To log into Maintain, type L[ogin]after Maintain's GV prompt. Now type your name and registry and press the carriage return. Maintain will prompt you for your password, each character of which will appear as an asterisk on the screen. Grapevine maintains its own copy of your password, and the password you give here must be the one Grapevine knows. If you give your password successfully, Maintain will inform you that you are logged in. Once you are logged in, you may proceed to issue other Maintain commands. If you cannot log in, because you have forgotten your password or for some other reason, then you will have to ask an administrator for your registry to issue a Change Password command.

Maintain automatically logs in the current user when it starts up, so you only need to issue the Login command if you wish to change your identity for some reason.

The ? Command

To obtain a list of Maintain commands, type a question mark after the GV prompt. You will see that the commands are Add Friend, Add Member, Add Owner, Change Password, Change Remark, Login, Quit, Remove Friend, Remove Member, Remove Owner, Type Entry, Type Members, and ^YžEnter Lisp. Each of these commands is described below.

The Type Entry Command

The Grapevine system has a *name data base* of entries giving information about groups and individuals. Groups represent distribution lists and some other groups that are unimportant to casual users, such as access control lists. Individuals represent human users and server computers. There are also ‘‘abstract users’’ that are syntactically individuals but actually name a group, such as LafiteSupport.PA.

To read an entry in the data base, type T[ype] E[ntry] after the GV prompt. Maintain will ask you for the name of the group or individual whose entry you want to read, then print the entry. When it is finished, it will return you to the GV prompt.

An entry for a group has four components: the *remark,* the number of *members,* a list of *owners,* and a list of *friends.* The remark describes the purpose of the group; for example, the remark for the distribution list Books^.PA is ‘‘Resource for Readers (primarily reviews).’’

The members are the individuals or other groups contained in the group. To obtain a list of members, use the Type Members command (described below).

The owners are the people empowered to add and remove owners, friends, and members for the group. They can also set and change the remark.

The friends are the people empowered to add and remove their own names to and from the group. If the list of friends is None, no one but the owners can add members to the list. If the list is an asterisk, anyone on the Grapevine system can add himself or herself to the list. If the list is of the form *. *Registry,* anyone in that registry can add himself or herself to the list, and so on.

An entry for an individual has two components: connect site and mailbox site(s).

The Type Members Command

To find out who the members of a distribution list are, type T[ype] M[embers] after the GV prompt. Maintain will fill in the words ‘‘of group’’ followed by a colon. Type the name of the distribution list after the colon and press the carriage return. Maintain will then print the list of members. If it is a very long list, your executive window may fill up, stop printing the list, and become highlighted in reverse video, a sign for you to hit the space bar to get the listing going again.

Some members of a group may be groups themselves. To see the members of those groups, you can use extra Type Members commands. If you use Type Members for the special group Groups.*Registry,* you will obtain a list of all distribution lists for that registry.

When Maintain is through printing the members of a distribution list, it will repeat the GV prompt.

The Add Member Command

You can add yourself (or another person if you are an owner) to a distribution list by typing A[dd] M[ember] [name] *YourName.Registry* after the GV prompt. Maintain will print ‘‘to group’’ after which you should fill in the name of the distribution list to which you want to add your name. If Maintain prints ‘‘done,’’ you have successfully added yourself to the list. If it prints ‘‘NoChange,’’ you were already a member of the list. If it prints ‘‘NotAllowed,’’ you should send a message to the owners of the list, asking them to add your name.

The Remove Member Command

You can remove yourself (or another person if you are an owner) from a distribution list by using the Remove Member command. This works in the same way as the Add Member command.

The Add Owner Command

Use the Add Owner command to add an owner to a distribution list. This is not allowed unless you yourself are an owner. Otherwise, this works in the same way as the Add Member command.

The Remove Owner Command

Use the Remove Owner command to remove an owner from a distribution list. This is not allowed unless you yourself are an owner. Otherwise, this works in the same way as the Remove Member command.

The Add Friend Command

Use the Add Friend command to add a friend to a distribution list. You cannot do this unless you are an owner. Otherwise, this works in the same way as the Add Member command.

The Remove Friend Command

Use the Remove Friend command to remove a friend from a distribution list. You cannot do this unless you are an owner. Otherwise, this works in the same way as the Remove Member command.

The Change Remark Command

The remark is the description of the distribution list that Maintain prints when you use the Type Entry command. To change the remark, type C[hange] R[emark] after the GV prompt. Maintain will prompt you for the name of the group, then ask you to type the new remark followed by a carriage return. You cannot change the remark for a list unless you are an owner.

The Change Password Command

You can change your Grapevine password using Maintain's Change Password command. Type C[hange] P[assword] after the GV prompt. Maintain will respond with ''for individual.'' Now type in your name and registry. Maintain will prompt with ''to be.'' Type in your new password, which will appear as a series of asterisks. Maintain will ask you to confirm your new password by typing it again. When you terminate this word with a carriage return or a space, Maintain will store it in the Grapevine data base. Be very careful when typing your new password, because if you make a mistake and can't reproduce the password later you will have to obtain the help of an administrator for your registry to correct it.

After returning to the executive, you will need to reinvoke the Login command (see above) to continue using Lafite to get and send mail.

The ^Y Command

If you want to drop into Lisp temporarily, then return to Maintain, type control-Y at the GV prompt. Maintain will fill in ''Enter Lisp'' and return you to the regular executive window prompt. To go back to Maintain, type OK.

The Quit Command

To leave Maintain, type Q[uit] after the GV prompt. Maintain will ask you to confirm this command with a carriage return.

If You're on Both Networks

Because the Grapevine and NS systems are totally separate, you must have a registered name and password to send mail on each system you are using. You can have two different passwords, but it is advisable to use the same one because otherwise you will be prompted to log in again each time you switch systems. You can also call (LOGIN 'GV::) yourself to set your Grapevine log-in, and (LOGIN 'NS::) to set an NS log-in, apart from your default (Grapevine) log-in.

Lafite Modes

When you use Lafite on the Grapevine system, Lafite is in Grapevine mode. When you use Lafite on the NS system, it is in NS mode. Mail sent to you on a particular system can be accessed, answered, and forwarded only if Lafite is in that system's mode.

If you have loaded both NS and Grapevine protocol handlers (NS Mail and MailClient), Lafite needs to know which mode to operate in. The first time you start Lafite after loading a new Lafite or a full sysout, you must set the mode explicitly. Until you do this, the Lafite status window reads ‘‘Mode Not Set.’’ After that, Lafite starts up in the same mode it was in when you quit Lafite. To change modes after you have intialized Lafite, use the middle mouse button to choose Quit from the status menu. Another menu containing several options will appear; one is GV Mode and another is NS Mode. Select the mode you want with the left mouse button. The center region of your status window reflects the mode you select. If you are in Grapevine mode, it says Lafite (GV). If you are in NS mode, it says Lafite (NS). You can also find out what mode you are in by typing (LAFITEMODE) in your executive window, or change modes by typing (LAFITEMODE 'GV) or (LAFITEMODE 'NS). If you always want to start Lafite in the same mode, you can set the global variable LAFITEMODEDEFAULT to the value GV or NS, in which case Lafite will initialize itself in that mode.

At sites with multiple Lafite modes, the values of the global variables LISPSUPPORT, LAFITESUPPORT, and TEDITSUPPORT are association lists in the form ((*Mode1* ‘‘*Address1*’’) (*Mode2* ‘‘*Address2*’’)---). Because the Lisp Report, Lafite Report, and TEdit Report forms are currently all sent to Grapevine addresses, they cannot be used at all in NS mode.

From Grapevine to NS and Vice Versa

Without switching modes, you can send mail from a Grapevine to an NS address or vice versa through a *mail gateway.* The gateway provides pseudoregistries and pseudodomains for mail addressed to the other kind of network. (A list of the special registries and the domains they correspond to is stored on {Indigo}<Registrar>GV>GV-NS-Mapping.Txt.) A single message can contain both Grapevine and NS addresses.

When Lafite is in Grapevine mode, you can send mail to the NS address *Name:Domain:Organization* by addressing it to ‘‘*Name:Domain:Organization*’’.NS (the double quotes are needed to keep the colon and any spaces in the name from being misparsed). For convenience, the domains PARC, OSBU North, and OSBU South can also be addressed as the registry of the same name, with the spaces removed. Thus, to send mail to Jones:PARC:Xerox, send it to Jones.PARC; to send mail to Oscar:OSBU North:Xerox, send it it Oscar.OSBUNorth.

When Lafite is in NS mode you can send mail to the GV address *Name.Registry* by addressing it to *Name.Registry:*GV. For convenience, the registries PA and ES can be addressed directly as NS domains. Thus, to send mail to Jones.PA, send it to Jones:PA.

Conversion of Grapevine Distribution Lists for NS Users. Until recently, people who used only NS mail could not add themselves to the many Grapevine public distribution lists. To address this problem, a new naming structure is being applied to general-interest public distribution lists in the PA, ES, and X registries. (Most organization and project lists, or any lists that might be used for access control, will not be converted to the new naming structure.) The new structure allows both NS and Grapevine mail users to add themselves to, broadcast to, and receive messages from these lists. Grapevine users will have to change the way they add themselves to lists. NS users will have to change both the way they send messages to lists that currently reside in Grapevine and the way they add themselves to lists.

To enable the new distribution list structure, new NS domains have been established in a one-to-one correspondence with the ES, PA, and X Grapevine registries. ‘‘All Areas:Xerox’’ is the NS domain analogous to the X registry. ‘‘PA Area:Xerox’’ is the NS analog of the geographically based registry PA, and ‘‘ES Area:Xerox’’ is analogous to ES. After the restructuring is implemented for a distribution list in, for example, the X registry, Grapevine users will continue to broadcast to *List*^.X, but will add themselves to *List*-GV^.X. NS users will broadcast to *List:*All Areas, but will add themselves to *List*-NS:All Areas. Either form of broadcast will reach both the Grapevine members (*List*-GV^.X) and the NS members (*List*-NS:All Areas:Xerox).

The conversion process will take several months. Distribution lists will be converted upon requests from their owners. If you own a distribution list and wish to request conversion, retrieve {USC:OSBU South:Xerox}<Public USC>DLRequest.Form, fill out the form, and send it to the appropriate address, according to the table below.

Registry	Grapevine User	NS User
ES lists	UserAdministration.ESArea	UserAdministration:ES Area:Xerox
PA lists	UserAdministration.PAArea	UserAdministration:PA Area:Xerox
X lists	UserAdministration.AllAreas	UserAdministration:AllAreas:Xerox

As soon as a distribution list of which you are a member is converted to the new naming structure, you will receive notice of the conversion. If you are a Grapevine member of *List*^.X, you will be added to *List*-GV^.X and removed from *List*^.X. If you are a Grapevine user and want to add yourself to a distribution list of which you are not currently a member, use Maintain to attempt to add yourself to *List*-GV^.X. If Maintain complains that this list does not exist, then add yourself to *List*^.X. You should continue to address messages to *List*^.X.

If you are primarily an NS user and are a member of Grapevine *List*^.X, then when that distribution list is converted to the new structure, you will be moved to *List*-GV^.X. Then you should ask your system administrator to add you to *List*-NS:All Areas:Xerox. To make sure you don’t miss any mail, wait three days after you start receiving two copies of messages sent to *List*^.X, then remove yourself from *List*-GV^.X with Maintain. To add yourself to a new distribution list, ask your system administrator to add you to *List*-NS:All Areas:Xerox. If this list does not exist, then add yourself to *List*^.X using Maintain. You should broadcast messages to *List:*All Areas:Xerox.

One key point to remember is to never send messages to a list with a -GV or -NS suffix. If you do, you will not reach both NS and Grapevine members. Another key point is, never broadcast to both *List*^.X and *List*:AllAreas, or everybody will get two copies of the message.

A more detailed description of the conversion process is filed on {USC:OSBU South:Xerox}<Public USC>DLConversion.Doc and {Indigo}<Registrar>GV>DLConversion.Doc.

Lafite-Related Lisp Users’ Packages

As of this writing, Xerox employees have access to four Lafite-related Lisp Users’ packages that have not yet been released to customers. The packages are Mailomat, Mail Sorter, Mail Cards, and Mail Reader. Mailomat enables you to automatically retrieve mail for one or multiple users for whom you have passwords. The program is filed on {Eris}<Lisp>Koto>LispUsers>Mailomat.DCOM; the documentation is on {Eris}<Lisp>Koto>LispUsers> Mailomat.TEdit. Mail Sorter will automatically sort mail into folders according to a set of rules you have specified. The code and documentation are on {QV}<Pimp>MailSorter.DCOM and {QV}<Pimp>MailSorter.TEdit, respectively. Mail Cards integrates the Note Cards environment with the Lafite mail system; you can place Lafite messages in specialized note cards, which are then placed in a Note Cards network. The code is stored on {QV}<Pimp>MailCards.DCOM and {QV}<Pimp>MailCardsLafiteInterface.DCOM; the documentation, on {QV}<Pimp>MailCards.TEdit. Lastly, the Mail Reader uses a voice synthesizer to read your mail to you over the telephone. You do not need special code to use this package, because you enter all commands by pressing the buttons on a touch-tone telephone. The Mail Reader's documentation is stored on {QV}<Speech>Speech> MailReader>.MailReader.TEdit.

All the abovementioned file names are subject to change. If you can't find a specific package, ask a release master or a network administrator for a pointer.

From Laurel to Lafite

Previous users of the Laurel mail program will find Lafite to have a similar user interface. Some of the ways in which it differs from Laurel are the following:

Laurel can only access mail folders on the local disk; Lafite can access folders on remote file servers. Thus, it is not necessary to transfer mail folders back and forth between your local disk and your file servers.

Laurel can only ‘‘browse’’ one mail folder at a time; Lafite can have several mail folders ‘‘opened’’ at the same time. Using the Interlisp window system, you can view the tables of contents of many mail folders and refer to messages in them independently.

You may have multiple windows displaying messages and multiple windows for sending messages and you may move text freely among them.

Messages can be selected in the table of contents by clicking *anywhere* within the summary line, unlike in Laurel, where you must select at the left end.

The Browse Laurel File Command

Lafite can read mail files written by the Laurel and Hardy mail programs; the files it writes are in Laurel format.

To browse a Laurel file that was produced by the Laurel mail reader version 6.1 or later, use the Browse Laurel File command from the middle-button Browse menu. Laurel 6.1 files are almost the same as Lafite files, but contain some line-formatting information that is stripped out by this command. After you have applied this command once to a file, you can subsequently browse the file with the normal Browse command (unless you use Laurel on it again, of course). This command is not intended for repeated use, but simply to make mail files built by Laurel more pleasing to browse. This command has the side effect of destroying any TEdit-formatted messages, so it should be used with care.

GLOSSARY OF TECHNICAL TERMS

This glossary covers hardware and software terms not specific to Lafite that are used in this manual.

argument Element of a Lisp function that specifies what that function operates on. For example, when you call (LOAD *'FILENAME*), *FILENAME* specifies what file is to be loaded. Note the use of parentheses and the single quote mark—the quote causes *FILENAME* to be taken literally, rather than as the name of a variable to be evaluated.

association list A list of lists that associates a key (usually an atom) with a value. The first element of each sublist is the key; the rest of the sublist is a value.

atom Any continuous string of characters, numbers, letters, or combinations thereof (except some prohibited characters such as parentheses). An atom is the smallest structure in Lisp.

back up To duplicate files to use in case the originals are destroyed in a hardware or software failure.

bit map A representation of any graphical entity as a sequence of bits.

break A state entered by Lisp during error processing that allows you to recover from the error by typing commands in a break window. If you don't know what to do with a break, type ^ after the prompt to abort the operation, then start over.

buffer A temporary storage area in a computer.

bug Any error, malfunction, or problem with hardware or software that produces unexpected or unintended results.

cache To save or store intermediate results to avoid having to recompute them.

caret A blinking pointer indicating where keyboard characters will appear when typed.

case sensitive Sensitive to case, that is, upper- and lowercase letters have different meanings.

Clearinghouse A Xerox Network service that provides a directory function within an internetwork, allowing all other system components to locate requested resources and other registered objects. It is implemented as a distributed system.

cursor A small picture (usually an arrow) on the display that tracks the motion of the mouse.

data base A collection of data organized for rapid search and retrieval.

DEdit The Interlisp-D editor for programming code.

default An action taken (or value specified) unless another action is specified by the user.

directory A set of one or more files that are stored together in the same place on a device.

executive window The window the blinking caret is in when you first start up Lisp on the machine. Note that if this and any other window overlap, the executive window dominates (i.e., covers) the other while control is in the executive window. To transfer control to another window (that accepts type-in), click the mouse in it, which usually also brings the window to the top.

extension The second part of a file name, usually used to indicate the type of file. It is separated from the first part of the file name (which indicates what the file contains) by a period. For example, the extension for Lafite mail folders is Mail, so a typical file name might be Active.Mail.

field A part of a message header, preceded by a field name and a colon. Contains information identifying the sender, recipients, subject, or other information of interest either to the users or to the mail system.

file server A computer on the network that provides a file storage and retrieval service.

format To produce in a specified form; the layout of a document, etc.

function A Lisp procedure that carries out a series of steps to produce some result. A function has a name and zero or more arguments on which it does its work.

global variable A variable accessible globally to all loaded programs, and whose value is usually not changed, except explicitly by the user (e.g., in an init file). Global variables are often used to personalize some aspect of the behavior of a program.

hard copy The physical copy of an on-screen document.

host Any machine on a network. Often used to refer specifically to a machine that provides a network service, such as filing.

icon A pictorial representation, usually of a shrunken window. An icon can be moved about on the screen by clicking on it with the left mouse button, and can be expanded by clicking on it with the middle mouse button.

intialization (init) file A file that is loaded when an Interlisp sysout is first started, and which usually customizes your Lisp environment according to your tastes and the idiosyncracies of your site. The usual arrangement is to have a site initialization file, which supplies information common to all users at your site (e.g., the name of your printer or directory search paths), and a personal initialization file, which supplies information about how you personally like your environment set up (e.g., where on the screen you like your Lafite windows).

keyword A significant word from a title or document used as an index to content.

mail server A computer that stores and distributes mail.

menu A collection of text strings, buttons, or icons on a display screen generally used to present a set of possible actions.

microcode The microinstructions of a computer, especially a microprocessor.

mode A particular functioning arrangement or condition of a computer.

network An interconnection, by some communications medium, of several computers, allowing them to communicate and share resources. Sometimes also used to refer to the medium itself, e.g., a coaxial cable in the case of the Ethernet.

package An Interlisp-D tool, or program, designed to help you carry out some complex procedure or task. Lafite is a package that enables you to read and send mail. A package must be loaded into your environment before you can use it.

parse To analyze something into its constituent parts, either to expose the structure or extract information. Specifically used to refer to the action of locating the message boundaries in a mail folder, or extracting fields from a message header (e.g., to discover who the message is addressed to).

position As a Lisp data type, a pair of display coordinates, in the form (*X*Position .*Y*Position). If you want to know the coordinates of a particular spot on the screen, type (GETPOSITION) and then click the mouse over the desired spot.

protocols Standards specifying how machines exchange information over a network.

random access Access to arbitrary parts of a file in no particular order (contrast with sequential access, which is accessing a file from the start sequentially through to the end).

region As a Lisp data type, a set of numbers describing a rectangular region of the display, in the form (*LEFTCOORDINATE BOTTOMCOORDINATE WIDTH HEIGHT*). If you want to know the region corresponding to any particular rectangular area on the screen, type (GETREGION) and then shape the rectangle as desired.

scavenge To try to fix messages that are not in the proper format.

search path List of directories to be searched for a file in the order given.

stream A programming abstraction that links your display to your message file.

string A sequence of characters. As a type of Lisp data, always entered in double quotes.

sysout A frozen version of an Interlisp-D environment. It contains all the information needed to initialize virtual memory when Interlisp-D is started.

time out To stop waiting for a user response, or to sever a network connection because of inactivity on the part of one of its participants.

value A word or a number that a variable is set to.

virtual memory Large working space on a local disk within which the machine runs Lisp.

GLOSSARY OF TECHNICAL TERMS

This glossary covers hardware and software terms not specific to Lafite that are used in this manual.

argument Element of a Lisp function that specifies what that function operates on. For example, when you call (LOAD *'FILENAME*), *FILENAME* specifies what file is to be loaded. Note the use of parentheses and the single quote mark—the quote causes *FILENAME* to be taken literally, rather than as the name of a variable to be evaluated.

association list A list of lists that associates a key (usually an atom) with a value. The first element of each sublist is the key; the rest of the sublist is a value.

atom Any continuous string of characters, numbers, letters, or combinations thereof (except some prohibited characters such as parentheses). An atom is the smallest structure in Lisp.

back up To duplicate files to use in case the originals are destroyed in a hardware or software failure.

bit map A representation of any graphical entity as a sequence of bits.

break A state entered by Lisp during error processing that allows you to recover from the error by typing commands in a break window. If you don't know what to do with a break, type ^ after the prompt to abort the operation, then start over.

buffer A temporary storage area in a computer.

bug Any error, malfunction, or problem with hardware or software that produces unexpected or unintended results.

cache To save or store intermediate results to avoid having to recompute them.

caret A blinking pointer indicating where keyboard characters will appear when typed.

case sensitive Sensitive to case, that is, upper- and lowercase letters have different meanings.

Clearinghouse A Xerox Network service that provides a directory function within an internetwork, allowing all other system components to locate requested resources and other registered objects. It is implemented as a distributed system.

cursor A small picture (usually an arrow) on the display that tracks the motion of the mouse.

data base A collection of data organized for rapid search and retrieval.

DEdit The Interlisp-D editor for programming code.

default An action taken (or value specified) unless another action is specified by the user.

directory A set of one or more files that are stored together in the same place on a device.

executive window The window the blinking caret is in when you first start up Lisp on the machine. Note that if this and any other window overlap, the executive window dominates (i.e., covers) the other while control is in the executive window. To transfer control to another window (that accepts type-in), click the mouse in it, which usually also brings the window to the top.

extension The second part of a file name, usually used to indicate the type of file. It is separated from the first part of the file name (which indicates what the file contains) by a period. For example, the extension for Lafite mail folders is Mail, so a typical file name might be Active.Mail.

field A part of a message header, preceded by a field name and a colon. Contains information identifying the sender, recipients, subject, or other information of interest either to the users or to the mail system.

file server A computer on the network that provides a file storage and retrieval service.

format To produce in a specified form; the layout of a document, etc.

function A Lisp procedure that carries out a series of steps to produce some result. A function has a name and zero or more arguments on which it does its work.

global variable A variable accessible globally to all loaded programs, and whose value is usually not changed, except explicitly by the user (e.g., in an init file). Global variables are often used to personalize some aspect of the behavior of a program.

hard copy The physical copy of an on-screen document.

host Any machine on a network. Often used to refer specifically to a machine that provides a network service, such as filing.

icon A pictorial representation, usually of a shrunken window. An icon can be moved about on the screen by clicking on it with the left mouse button, and can be expanded by clicking on it with the middle mouse button.

intialization (init) file A file that is loaded when an Interlisp sysout is first started, and which usually customizes your Lisp environment according to your tastes and the idiosyncracies of your site. The usual arrangement is to have a site initialization file, which supplies information common to all users at your site (e.g., the name of your printer or directory search paths), and a personal initialization file, which supplies information about how you personally like your environment set up (e.g., where on the screen you like your Lafite windows).

keyword A significant word from a title or document used as an index to content.

mail server A computer that stores and distributes mail.

menu A collection of text strings, buttons, or icons on a display screen generally used to present a set of possible actions.

microcode The microinstructions of a computer, especially a microprocessor.

mode A particular functioning arrangement or condition of a computer.

network An interconnection, by some communications medium, of several computers, allowing them to communicate and share resources. Sometimes also used to refer to the medium itself, e.g., a coaxial cable in the case of the Ethernet.

package An Interlisp-D tool, or program, designed to help you carry out some complex procedure or task. Lafite is a package that enables you to read and send mail. A package must be loaded into your environment before you can use it.

parse To analyze something into its constituent parts, either to expose the structure or extract information. Specifically used to refer to the action of locating the message boundaries in a mail folder, or extracting fields from a message header (e.g., to discover who the message is addressed to).

position As a Lisp data type, a pair of display coordinates, in the form (*X*Position .*Y*Position). If you want to know the coordinates of a particular spot on the screen, type (GETPOSITION) and then click the mouse over the desired spot.

protocols Standards specifying how machines exchange information over a network.

random access Access to arbitrary parts of a file in no particular order (contrast with sequential access, which is accessing a file from the start sequentially through to the end).

region As a Lisp data type, a set of numbers describing a rectangular region of the display, in the form (*LEFTCOORDINATE BOTTOMCOORDINATE WIDTH HEIGHT*). If you want to know the region corresponding to any particular rectangular area on the screen, type (GETREGION) and then shape the rectangle as desired.

scavenge To try to fix messages that are not in the proper format.

search path List of directories to be searched for a file in the order given.

stream A programming abstraction that links your display to your message file.

string A sequence of characters. As a type of Lisp data, always entered in double quotes.

sysout A frozen version of an Interlisp-D environment. It contains all the information needed to initialize virtual memory when Interlisp-D is started.

time out To stop waiting for a user response, or to sever a network connection because of inactivity on the part of one of its participants.

value A word or a number that a variable is set to.

virtual memory Large working space on a local disk within which the machine runs Lisp.

Message Marks

? Means that a message has not yet been displayed.

a Means that a message has been answered.

f Means that a message has been forwarded.

h Means that a message has been hard-copied.

m Means that a message has been moved to another folder.

Note: Only one mark is present at a time; each mark replaces the previous one.

Lafite Commands

Status Window Commands

Browse

Pops up a menu of your mail folders. Selecting Browse with the middle button brings up another menu with the additional commands:

Browse Laurel File

Browses a file that was produced by Laurel.

Forget Folder

Removes a folder from the list of known mail folders.

Forget Message Form

Removes a message form from the list of known message forms.

Send Mail

Brings up a message composition window. If Send Mail is selected with the middle button, a menu is presented with the following choices:

Lisp Report

Provides a message template to report an Interlisp bug or make a suggestion.

Lafite Report

Provides a message template similar to the Lisp Report but sent to Lafite maintainers.

TEdit Report

Provides a message template similar to the Lisp Report but sent to TEdit maintainers.

Saved Form

Prompts for a form name.

Standard Form

Provides an empty message template.

Quit

Stops Lafite and closes all browser windows. Selecting the Quit command with the middle button brings up a menu of status-changing commands:

Restart

Turns Lafite off, then on again.

Browser Window Commands

Display

Displays the selected message.

Delete

Deletes the selected message.

Undelete

Undeletes the selected message.

Answer

Constructs an Answer form for the selected message.

Forward

Constructs a Forward form for the selected message.

Hardcopy

Prints the selected message.

Move To

Pops up a menu of known mail folders and moves the selected message to the chosen folder.

Update

Transmits the changes that you made to your mail folder to the physical mail file. Selecting Update with the middle mouse button brings up a menu with the commands:

Write Out Changes Only

Makes the browser and the mail file completely consistent with each other.

Expunge Deleted Messages

Removes all messages marked deleted, in addition to making the browser and the mail file completely consistent with each other.

Get Mail

Brings new mail into the folder.

Message Composition Window Commands

Deliver

Sends your message. During delivery, the menu atop the message window changes into a single item, Abort; if you click on this item, the delivery is aborted.

Save Form

Asks you for a file name on which to save this message for later use as a message form.

QUICK-REFERENCE SECTION

Message Marks

? Means that a message has not yet been displayed.

a Means that a message has been answered.

f Means that a message has been forwarded.

h Means that a message has been hard-copied.

m Means that a message has been moved to another folder.

Note: Only one mark is present at a time; each mark replaces the previous one.

Lafite Commands

Status Window Commands

Browse

Pops up a menu of your mail folders. Selecting Browse with the middle button brings up another menu with the additional commands:

Browse Laurel File

Browses a file that was produced by Laurel.

Forget Folder

Removes a folder from the list of known mail folders.

Forget Message Form

Removes a message form from the list of known message forms.

Send Mail

Brings up a message composition window. If Send Mail is selected with the middle button, a menu is presented with the following choices:

Lisp Report

Provides a message template to report an Interlisp bug or make a suggestion.

Lafite Report

Provides a message template similar to the Lisp Report but sent to Lafite maintainers.

TEdit Report

Provides a message template similar to the Lisp Report but sent to TEdit maintainers.

Saved Form

Prompts for a form name.

Standard Form

Provides an empty message template.

Quit

Stops Lafite and closes all browser windows. Selecting the Quit command with the middle button brings up a menu of status-changing commands:

Restart

Turns Lafite off, then on again.

Browser Window Commands

Display

Displays the selected message.

Delete

Deletes the selected message.

Undelete

Undeletes the selected message.

Answer

Constructs an Answer form for the selected message.

Forward

Constructs a Forward form for the selected message.

Hardcopy

Prints the selected message.

Move To

Pops up a menu of known mail folders and moves the selected message to the chosen folder.

Update

Transmits the changes that you made to your mail folder to the physical mail file. Selecting Update with the middle mouse button brings up a menu with the commands:

Write Out Changes Only

Makes the browser and the mail file completely consistent with each other.

Expunge Deleted Messages

Removes all messages marked deleted, in addition to making the browser and the mail file completely consistent with each other.

Get Mail

Brings new mail into the folder.

Message Composition Window Commands

Deliver

Sends your message. During delivery, the menu atop the message window changes into a single item, Abort; if you click on this item, the delivery is aborted.

Save Form

Asks you for a file name on which to save this message for later use as a message form.

1. INTRODUCTION

What Lafite Is

Lafite is an easy-to-use, display-oriented electronic message system. Like all electronic message systems, Lafite combines the advantages of the postal service and the telephone system. You can compose messages at leisure and in private, transmit them rapidly, and print copies of the messages you send and receive.

However, Lafite offers more than most electronic message systems. It uses the Interlisp-D window interface, which enables you to read and write many messages simultaneously while you are using other Interlisp systems. It is also fully integrated with the TEdit text editor, which allows you to write messages using all of TEdit's editing and formatting capabilities. In addition, Lafite enables you to create mail folders and file your messages in them and to create, save, and use special message forms.

What This Manual Is About

This manual is designed to help you use Lafite. It assumes you understand the basic principles of using your Xerox Lisp workstation and the Interlisp-D environment. If you need information on operating your workstation, refer to its user's manual. The *Friendly Dandelion Primer* and the *Interlisp-D Reference Manual* describe the Interlisp environment.

If you are a new Lafite user, you will find it helpful to skim through the manual, then return to the beginning and practice using Lafite's features in the order in which they are described. If you are an experienced Lafite user, you should just skim through the manual, then turn to the appropriate section whenever you need help. If you are a technically knowledgeable or adventurous user, you may want to know how to customize the system. Information on modifying Lafite's behavior is provided in chapter 13, ''Customizing Lafite.''

You can find some useful background information in chapter 2, ''Things to Know Before You Start.'' Chapter 3, ''Obtaining Lafite,'' tells you how to log in and how to load Lafite and the other software you need to use it. In chapter 4, ''Entering Lafite,'' you can find out how to access Lafite and how to use the highest-level Lafite window, the status window. You can learn how to read your mail in chapter 5, ''Reading Messages,'' and how to update your mail files in chapter 6, ''Deleting Messages and Updating the Mail Folder.'' Chapter 7, ''Printing Messages,'' tells you how to obtain hard copies of your messages, and chapter 8, ''Filing Messages,'' tells you how to organize them. You can find out how to compose and transmit Lafite mail in chapters 9 and 10, ''Writing Messages'' and ''Sending Messages.'' Chapter 11, ''Leaving Lafite,'' describes several ways you can exit the system.

If Lafite does not seem to be operating correctly, you can turn to chapter 12, ‘‘Troubleshooting Lafite Problems.’’ Chapter 13, ‘‘Customizing Lafite,’’ tells you how to change Lafite’s global variables (we have tried to make it accessible to nonprogrammers who are willing to do a little extra reading). And chapter 14, ‘‘Using Lafite-Related Lisp Library Packages,’’ contains user’s guides to the Lafite Find and Mail Scavenger Lisp Library packages.

You will find several different kinds of reference material at the end of this manual. Appendix A, ‘‘Using Lafite Courteously,’’ is a guide to message system etiquette. You can look up technical terms used in the manual in the glossary at the end of appendix A. After appendix A, you will find a quick-reference section giving brief summaries of Lafite’s commands and a chart of message marks. At the very end, there is an index that shows you where to find all the important information given in this manual.

2. Things to Know Before You Start

Lafite is a display-based, interactive program that manipulates a particular class of files called *mail files.* In essence, a mail file is just a set of messages, each of which is a piece of text formatted according to certain conventions. The details of these conventions are not of major interest to most users. Suffice it to say that messages have a *header* and a *body*. The header identifies the *sender* and *recipients* of the message as well as the *time* it was sent, a *subject,* and perhaps other information not directly related to the message's content. The body contains the content of the message. For each mail file, Lafite constructs and maintains a *table of contents* that summarizes the messages residing in the file. Mail files can be stored either on a file server that allows random access or on your own workstation's disk.

Lafite is an interface program that gives you access to the facilities provided by the Xerox Network Systems (NS) Mail Service. This system provides you with *in-boxes, mail drops,* and a *name data base.* An in-box is a place where messages sent to you are stored until you retrieve them. Each in-box resides on a *mail server,* which is a machine used for storing and distributing electronic mail. A mail drop is a service to which messages addressed to any user(s) of the system may be sent. The mail transport system sorts out the recipients and delivers messages to their individual in-boxes. The name data base holds the names of *registered users* of the message system along with named lists of such users.

Lafite retrieves inbound mail from your in-boxes on one or more mail servers. The messages are retrieved into a *mail folder,* which is usually called Active.Mail (henceforth referred to as the Active Mail folder). You can then open a *browser* onto the mail folder, which allows its contents to be read, replied to, forwarded, deleted, moved into other folders, printed, and so forth.

3. Obtaining Lafite

What Programs You Need

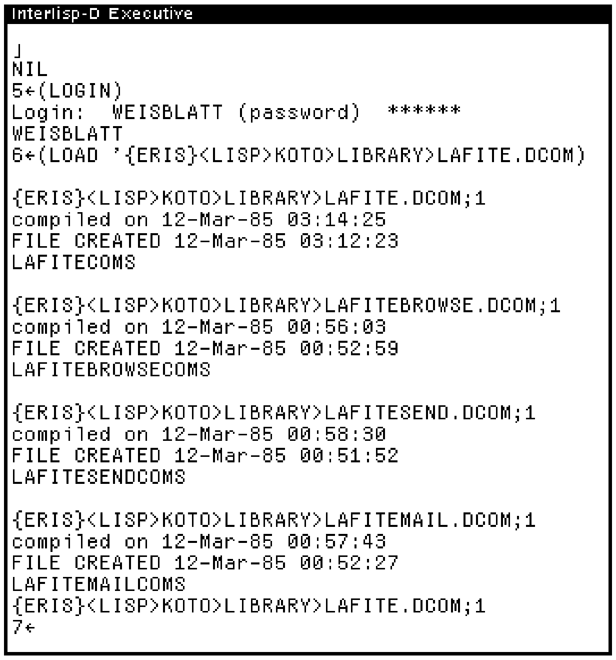
Before you run Lafite, two Lisp Library programs must be loaded into your workstation's virtual memory. These are Lafite itself and NS Mail. NS Mail contains an implementation of the *protocols,* or information interchange standards, that enable Lafite to communicate with NS mail servers.

In addition, you may want to load two optional but useful programs, Lafite Find and Mail Scavenger. Lafite Find facilitates searching for messages contained in a mail folder. Mail Scavenger helps to restore the internal structure of a mail file in the unlikely event that it becomes damaged.

How to Load the Programs You Need

To load Lafite or another program from a floppy disk, type (LOAD '{FLOPPY}*FILENAME*.DCOM) at the prompt in your Interlisp-D Executive window. However, if a program is frequently used in your company it is probably stored on a file server rather than on a floppy disk. The following instructions apply to loading packages from a file server, with the words *FILESERVER* and *DIRECTORY* standing for the names of the actual file server and directory on which these programs are stored.

First, you must log into the system. To log in, type (LOGIN) at the prompt in the executive window (see figure 1). On the next line, the system will print the word Login, followed by a colon. Type your log-in name after the colon and press the carriage return. On the same line, the system will prompt you for your password. Type in your password (each letter will appear as an asterisk on the screen) and press the carriage return again. The system will print your name as a confirmation of your log-in.



```
Interlisp-D Executive

J
NIL
5←(LOGIN)
Login:  WEISBLATT (password)  ******
WEISBLATT
6←(LOAD '{ERIS}<LISP>KOTO>LIBRARY>LAFITE.DCOM)

{ERIS}<LISP>KOTO>LIBRARY>LAFITE.DCOM;1
compiled on 12-Mar-85 03:14:25
FILE CREATED 12-Mar-85 03:12:23
LAFITECOMS

{ERIS}<LISP>KOTO>LIBRARY>LAFITEBROWSE.DCOM;1
compiled on 12-Mar-85 00:56:03
FILE CREATED 12-Mar-85 00:52:59
LAFITEBROWSECOMS

{ERIS}<LISP>KOTO>LIBRARY>LAFITESEND.DCOM;1
compiled on 12-Mar-85 00:58:30
FILE CREATED 12-Mar-85 00:51:52
LAFITESENDCOMS

{ERIS}<LISP>KOTO>LIBRARY>LAFITEMAIL.DCOM;1
compiled on 12-Mar-85 00:57:43
FILE CREATED 12-Mar-85 00:52:27
LAFITEMAILCOMS
{ERIS}<LISP>KOTO>LIBRARY>LAFITE.DCOM;1
7←
```

Figure 1. The executive window. The user has logged in and loaded Lafite

You can now load Lafite. Type (LOAD '*{FILESERVER}<DIRECTORY>*LAFITE.DCOM) at the prompt. (If the DIRECTORIES variable in your initialization file contains a search path to the directory on which Lafite is stored, you can just type (LOAD 'LAFITE.DCOM)). The system will print the file's full name, then give the date the file was created. When it is finished loading the file, it will again print the file name and return you to the top-level prompt.

To load NS Mail, type (LOAD '*{FILESERVER}<DIRECTORY>*NSMAIL.DCOM).

To load Lafite Find, type (LOAD '*{FILESERVER}<DIRECTORY>*LAFITEFIND.DCOM).

Finally, to load Mail Scavenger, type (LOAD '*{FILESERVER}<DIRECTORY>*MAILSCAVENGE.DCOM).

4. Entering Lafite

You must log into the network before you can use Lafite. If you have not logged in already, do so now, following the instructions given in chapter 3.

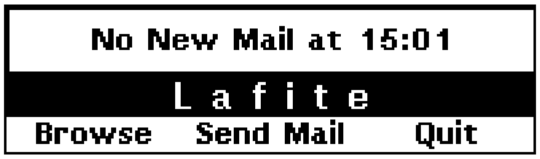
In order to obtain your mail, you must be an *authenticated user;* i.e., your name and password must be accepted by your mail service. When you start up Lafite, it obtains your name and password from the Lisp executive and submits them to the authenticator for verification. If the authentication fails, you are prompted to log in. Lafite will not allow you to send or receive messages until you have been authenticated.

You can start Lafite by typing one of two commands at the prompt in the executive window: (LAFITE) or (LAFITE 'ON). Lafite will automatically supply your Active Mail folder. You can look at another folder, say *MAILFOLDER*, by typing (LAFITE 'ON *'MAILFOLDER)*. You can also type (LAFITE 'ON NIL) to start Lafite without opening any mail folder, then later select a folder from the Browse menu after you enter Lafite; see chapter 5, ''Reading Messages.'' You can also write and send (but not read, move, or otherwise manipulate) Lafite messages by selecting SendMail from the background menu; see chapter 9, ''Sending Messages.''

The Status Window

When you start Lafite, two windows will appear on your workstation's screen: the *status window* and the *browser window* (the latter will not appear if you typed (LAFITE 'ON NIL)). The other two important types of Lafite windows are the *message display window* and the *message composition window*. The browser and message display windows are fully discussed in chapter 5, ''Reading Messages,'' and the message composition window is covered in chapter 9, ''Writing Messages.''

The Lafite status window contains a region for Lafite status information and a small, fixed menu of commands (see figure 2). When you first start Lafite, the status region informs you that Lafite is initializing and displays the time. This message quickly disappears, to be followed by either ''New Mail for *YourName*'' or ''No New Mail at *Time.*'' Your in-boxes are checked for new mail approximately every five minutes; Lafite reports in the status region if you have new mail. If you click in the status region, the background process wakes up and reports status immediately instead of waiting its normal interval.



No New Mail at 15:01

Figure 2. The status window, showing the status region, the title bar, and the menu of commands

After you have sent a message an additional window, your *out-box*, appears beneath the status window; it lists the dates and subjects of the messages you sent most recently. See chapter 10, ‘‘Sending Messages,’’ for more information on the out-box.

The status window (and all other Lafite windows except the message display window) has its own fixed menu of commands, which are printed in boldface type. You invoke a command by moving the mouse until the cursor points at that command, then clicking the left mouse button. When you hold down the button, the command will appear *inverted* (i.e., white letters on a black background). When you release the button, the command will appear *grayed* (i.e., black letters on a gray background). If, while holding down the mouse button, you change your mind about the command you intend to select, simply move the cursor off the inverted command name until it is restored to its normal state, then release the button. In some cases, clicking the middle mouse button instead of the left performs some different but related operation or brings up an additional menu; such commands are described below.

The menu region of the Lafite status window contains three commands—Browse, Send Mail, and Quit. The Browse command is discussed in chapter 5, ‘‘Reading Messages,’’ and the Send Mail command is discussed in chapter 10, ‘‘Sending Messages.’’ The Quit command is discussed in chapter 11, ‘‘Leaving Lafite.’’

If you are using this manual as a tutorial, ask several people in your company to send you messages before going on to chapter 5, ‘‘Reading Messages.’’

5. Reading Messages

The Browse Command

To access your mail from the Lafite status window, click the left mouse button on the Browse command. A menu will appear listing all your mail folders (see figure 3). Your new mail is conventionally stored in your Active Mail folder, and additional folders are added to the menu when you browse them or move messages into them. The creation of mail folders is discussed in chapter 8, ‘‘Filing Messages.’’

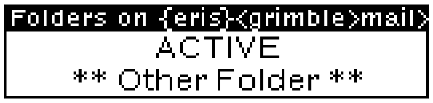
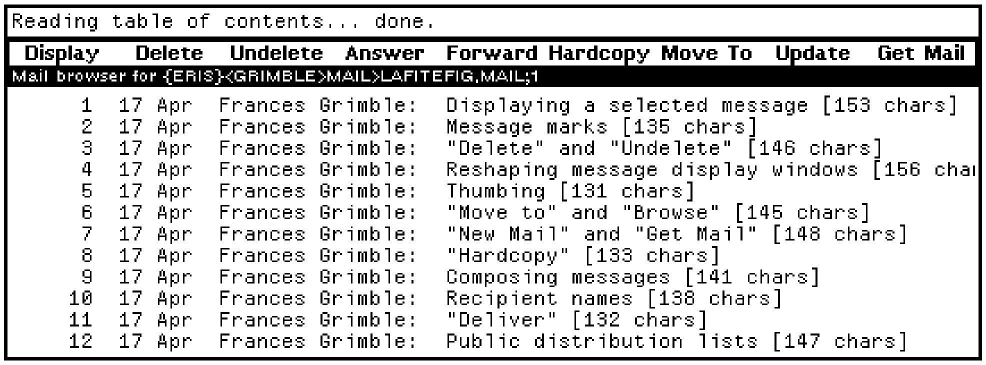


Figure 3. A folder menu, showing that the user has only one mail folder (the Other Folder item is used to create new folders)

To read the mail contained in a particular folder, such as Active Mail, click on the name of that mail folder with the left mouse button.

The Browser Window

Clicking on the name of a folder in the Browse menu opens a browser window onto that folder (see figure 4). The main part of the browser window displays a table of contents, which gives a one-line summary of each message, and a horizontal menu above the table of contents that contains commands that apply specifically to that folder. The topmost part of the browser window is the *prompt region.* In the prompt region, the system prints status information related to the browser and sometimes prompts you for information. The black *title bar* just above the table of contents labels the window.


Display Delete Undelete Answer Forward Hardcopy Move To Update Get Mail

```
Reading table of contents... done.
Mail browser for {ERIS}<GRIMBLE>MAIL>LAFITEFIG.MAIL;1
      1  17 Apr  Frances Grimble:  Displaying a selected message [153 chars]
      2  17 Apr  Frances Grimble:  Message marks [135 chars]
      3  17 Apr  Frances Grimble:  "Delete" and "Undelete" [146 chars]
      4  17 Apr  Frances Grimble:  Reshaping message display windows [156 cha
      5  17 Apr  Frances Grimble:  Thumbing [131 chars]
      6  17 Apr  Frances Grimble:  "Move to" and "Browse" [145 chars]
      7  17 Apr  Frances Grimble:  "New Mail" and "Get Mail" [148 chars]
      8  17 Apr  Frances Grimble:  "Hardcopy" [133 chars]
      9  17 Apr  Frances Grimble:  Composing messages [141 chars]
     10  17 Apr  Frances Grimble:  Recipient names [138 chars]
     11  17 Apr  Frances Grimble:  "Deliver" [132 chars]
     12  17 Apr  Frances Grimble:  Public distribution lists [147 chars]
```

Figure 4. A browser window, showing the prompt region, the menu of commands, the title bar, and the table of contents

The table of contents is an index of the messages in the mail folder being viewed. The first time you browse a folder Lafite has to construct the table of contents, which it reports in the prompt region as ‘‘Parsing folder.’’ Each summary line in the table of contents is numbered (the numbers are unique to that table of contents) and contains the date sent, the sender’s name, the subject, and the number of characters in the message.

Each entry can also have a single-character *message mark,* which is an additional tidbit of information about the message displayed in the summary line. A message originally comes with a question mark, meaning it is unexamined. Some Lafite commands change the mark automatically. For example, displaying a message removes the question mark. You can change the mark directly by selecting with the mouse in the narrow area immediately to the left of the message number, where the mark is printed. Simply click in the mark position and type in the new mark. The message mark is the only piece of information in the table of contents you can directly modify.

The Get Mail Command

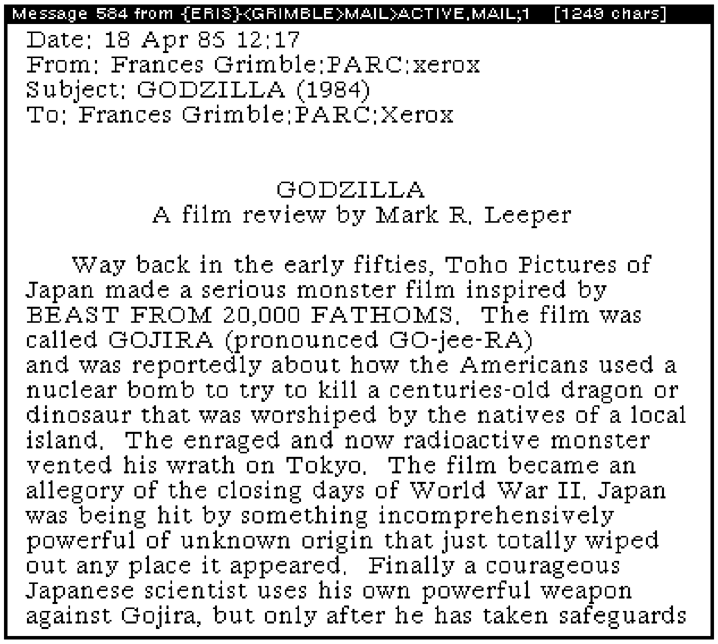
The table of contents in your folder does not display your new messages until Lafite retrieves them from your in-box. To instruct Lafite to move the contents of your in-box to your current mail file, point the cursor at the Get Mail command and click the left mouse button. The command then appears on a gray background and the prompt region reports on the activities of your mail server(s). When all the messages have been transferred, the gray background disappears and the prompt window tells you how many new messages you have. The table of contents is updated to include the new messages placed in your mail file; it is scrolled to display the first one.

Lafite automatically places a question mark next to each unread message in the table of contents (except for the ones you sent to yourself).

The Display Command and Selection Pointer

To read a message, you must first select its entry in the table of contents. After obtaining new mail from your in-box, Lafite automatically selects the first new message for you and places a *selection pointer* (a small black triangle pointing to the right) next to it. When you first browse a folder, Lafite selects the first unread message; if there are no unread messages in the folder, Lafite places the selection pointer next to the last message.

To display a selected message, point the cursor at the Display command in the browser window menu and click the left mouse button. A message display window will open and fill up with the message you selected (see figure 5). At this point, the question mark (if any) next to its entry in the table of contents disappears.



```
Message 584 from {ERIS}<GRIMBLE>MAIL>ACTIVE.MAIL;1    [1249 chars]
Date: 18 Apr 85 12:17
From: Frances Grimble:PARC:xerox
Subject: GODZILLA (1984)
To: Frances Grimble:PARC:Xerox


                    GODZILLA
          A film review by Mark R. Leeper

     Way back in the early fifties, Toho Pictures of
Japan made a serious monster film inspired by
BEAST FROM 20,000 FATHOMS.  The film was
called GOJIRA (pronounced GO-jee-RA)
and was reportedly about how the Americans used a
nuclear bomb to try to kill a centuries-old dragon or
dinosaur that was worshiped by the natives of a local
island.  The enraged and now radioactive monster
vented his wrath on Tokyo.  The film became an
allegory of the closing days of World War II. Japan
was being hit by something incomprehensively
powerful of unknown origin that just totally wiped
out any place it appeared.  Finally a courageous
Japanese scientist uses his own powerful weapon
against Gojira, but only after he has taken safeguards
```

Figure 5. A message display window, showing the title bar and the message display region. The message display window lacks the message composition window's prompt region, command menu, and TEdit pop-up menus; you cannot use it for editing

To examine the next message listed in the table of contents, click on Display again. The selection pointer will move to the next entry, and this message will replace the previous one in the message display window. Lafite will skip over deleted messages when advancing the selection pointer. To display a deleted message you must select it explicitly, then click on Display.

You may explicitly select any entry in the table of contents by moving the cursor into the line desired and clicking the left mouse button. Any existing selection pointers will be removed, and a new one will be placed at the indicated entry. You can have several messages selected at once. If you want to add a message to the current selection, click the middle mouse button instead of the left. To remove a message from the current selection, hold down the shift key while clicking any mouse button.

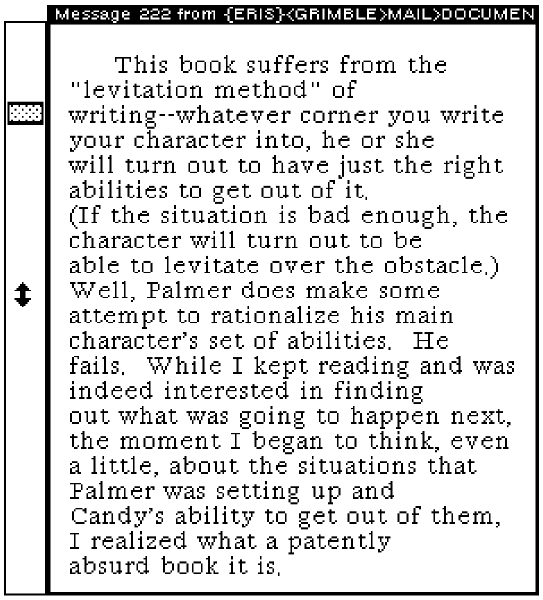
If you want to select several entries in a row, place the cursor at the first message you want to read and click the left mouse button. Then move to the last message you want to read and click the right mouse button. Lafite will place selection pointers next to all the messages in between (deleted messages are not included unless the control key—Props on an 1108 or 1186 workstation—is down). You may then read them in succession by clicking on Display each time you finish reading a message.

If you have selected more than one message, clicking on Display after you read the last selected message will cause the cursor to cycle back to the first selected message. The only way to break this cycle is to explicitly select a single message.

If you want several messages displayed on the screen at once, you must create additional message display windows. To do so, click on Display with the middle mouse button. Lafite will prompt you to create a window. After you have done so, the message you selected to display will appear in the new window, and the previously displayed message will remain in the original message display window. You may create as many message display windows as you wish.

Text Scrolling

Lafite browser and message display windows may contain more text than you can see at any one time. (This is also true of message composition windows, which are discussed in chapter 9, "Writing Messages.") You can view this text by using a scroll bar to scroll the window's contents (see figure 6). The Lafite browser window has two scroll bars, one just beyond the left margin and one just below the bottom margin. The message display window has only the left scroll bar. The left scroll bar is used for vertical scrolling, and the lower scroll bar is used for horizontal scrolling.


Message 222 from {ERIS}<GRIMBLE>MAIL>DOCUMEN

This book suffers from the
"levitation method" of
writing--whatever corner you write
your character into, he or she
will turn out to have just the right
abilities to get out of it.
(If the situation is bad enough, the
character will turn out to be
able to levitate over the obstacle.)
Well, Palmer does make some
attempt to rationalize his main
character's set of abilities. He
fails. While I kept reading and was
indeed interested in finding
out what was going to happen next,
the moment I began to think, even
a little, about the situations that
Palmer was setting up and
Candy's ability to get out of them,
I realized what a patently
absurd book it is.

Figure 6. A message display window showing a vertical scroll bar. The gray area represents the position of the text being read in relation to the message as a whole; the double-headed shape of the cursor means that the message is not currently being scrolled

A scroll bar can be brought into view by moving the mouse cursor onto that area of the screen. The cursor appears as a double-headed arrow until you begin scrolling, when it changes shape to point in the correct direction. The gray rectangle within the scroll bar represents the position of the text you are viewing in relation to the file as a whole. Within a vertical scroll bar, the left mouse button scrolls the contents of the window up, and the right mouse button scrolls them down. Within a horizontal scroll bar, the left mouse button scrolls the contents of the window to the left, and the right mouse button scrolls them to the right.

You can control the amount of text that is scrolled by moving your mouse cursor to different positions on the scroll bar. Scrolling with the cursor at the bottom of the vertical scroll bar moves the text about one window length at a time. Placing the cursor at the top of the bar moves the text about one line at a time. There is less variation in the amount of text you can scroll using a Lafite browser horizontal scroll bar, but placing the cursor at the left end scrolls the text by smaller amounts than placing it on the right. To produce continuous scrolling, hold down the mouse button; release it when the part of the file you want to read is shown.

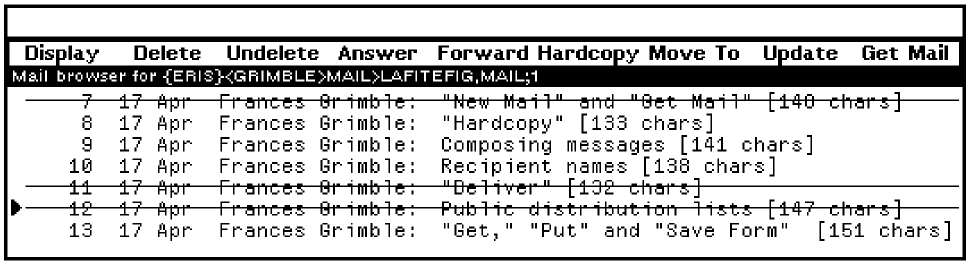
Text Thumbing

If you want to skip to a distant part of the text quickly, *thumbing* is better than scrolling. Thumbing is analogous to opening a book by placing your thumb at the approximate position of the section you want to read and pulling the book open at that point. To thumb the contents of a Lafite window, place the cursor in the part of the scroll bar that represents the part of the file you want to read. For instance, to reach the top of a file, place the cursor at the top of the vertical scroll bar. Then press the middle mouse button. The cursor will become a gray triangle pointing toward the window. When you release the mouse button, the text will move to show the part you want to read. If you haven't thumbed to quite the right place, you can find the text you want by scrolling.

6. Deleting Messages and Updating the Mail Folder

The Delete Command

After examining the messages in your mail file, you may wish to delete some of them. Select the messages you want to delete, then invoke the Delete command by pointing the cursor at it and clicking the left mouse button. Lafite will draw a black line through each deleted entry in the table of contents (see figure 7).



Mail browser for {ERIS}<GRIMBLE>MAIL>LAFITEFIG,MAIL;1

~~7 17 Apr Frances Grimble: "New Mail" and "Get Mail" [140 chars]~~
8 17 Apr Frances Grimble: "Hardcopy" [133 chars]
9 17 Apr Frances Grimble: Composing messages [141 chars]
10 17 Apr Frances Grimble: Recipient names [138 chars]
~~11 17 Apr Frances Grimble: "Deliver" [132 chars]~~
▶ ~~12 17 Apr Frances Grimble: Public distribution lists [147 chars]~~
13 17 Apr Frances Grimble: "Get," "Put" and "Save Form" [151 chars]

Figure 7. A browser window showing a table of contents from which three messages have been deleted

When processing newly arrived mail, you may wish to delete a message immediately after displaying it. In this case, Lafite will automatically select and display the next message (assuming you have not examined or deleted it).

The Undelete Command

Messages marked for deletion are not actually removed from your mail file at the time the Delete command is given. If you discover that you have inadvertently deleted a message you want to keep, select it, point the cursor at the Undelete command, and click the left mouse button. The line drawn through the table-of-contents entry of the selected message will be removed. You can select several messages and delete or undelete them all at once; see chapter 5, ‘‘Reading Messages,’’ for instructions on selecting messages.

The Update Command

With the exception of retrieving new mail, any change you make to a folder being browsed, such as deleting a message or changing a message mark, is not made permanent until you issue the Update command. That is, if you were to boot your machine, start up Lafite, and browse the same folder again, you would find it in the same state as when you last updated (or as when you last browsed it if you never updated). The Update command ensures that your mail file accurately reflects the contents of the browser, so that you could return to the same state if for any reason your machine failed and you had to start up again from scratch. It also provides the opportunity to reclaim the space occupied by deleted messages. Thus, it is important to regularly update your browsers.

When you click on the Update command, a menu appears listing your options: Write Out Changes Only or Expunge Deleted Messages (see figure 8). Ordinarily, you want to choose Expunge Deleted Messages, which, in addition to writing out all changes, permanently removes all deleted messages and renumbers the remaining ones. However, Expunge can take a long time if your folder is large and there are many ‘‘holes’’ in it caused by deleting messages, since Expunge has to compact the file. Thus, if you are processing a lot of mail and just want to be sure periodically that your changes are saved, with the intention to expunge the folder later, you can choose the Write Out Changes Only option, which writes out your changes without compacting out the deleted messages. This option is usually faster than Expunge.

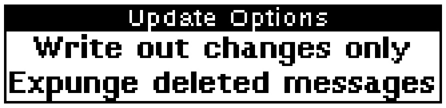


Figure 8. The update options menu

Other Ways to Update

When you attempt to close or shrink a browser window, using the Close or Shrink commands from the standard right-button window menu, Lafite will check whether you have made any changes to the folder, and if so, prompt you with a menu giving your updating options: Write Out Changes Only, Expunge Deleted Messages, or Don’t Update File. Choosing either of the first two options is the same as clicking on Update and choosing that option, followed by closing or shrinking the window, as you requested. If you choose Don’t Update File, then no update occurs—your changes are not saved.

If the only change you made to the folder was to retrieve new mail to it, or move messages into it from another folder (see chapter 8, ‘‘Filing Messages’’), then you are instead offered the options of Update Table of Contents or Just Close. The former is, in this case, all the updating that the folder requires.

You may find it convenient to keep the browser for your Active Mail folder always on your screen, shrinking it down to an icon to save screen space when you are not reading your mail (see figure 9). If you do this regularly, you may never need to use the Update command directly, instead combining the update with the shrinking of the window.

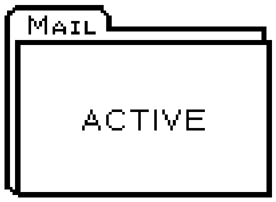


Figure 9. The icon for a shrunken mail folder, in this case Active Mail. You can shrink any Lafite window by placing the mouse cursor on the title bar of the window, pressing the right

button, and selecting Shrink from the menu of window commands that appears

7. Printing Messages

To print a copy of a message in your mail file, select it in the table of contents and click the Hardcopy command with the left mouse button. Several status messages will appear in the Interlisp-D prompt window reporting on the activities of your printer. When your message has been printed, its mark is changed to *h*. Lafite considers hard copy to be printed as soon as it has been accepted by the printer. There may be some additional delay before your message is actually printed.

You may select several messages to be printed together. Each message will be printed on a separate page (unless you have customized Lafite to batch printing requests).

You can print an unsent message from a message composition window using the right-button window menu's Hardcopy command; see chapter 9, ‘‘Writing Messages.’’

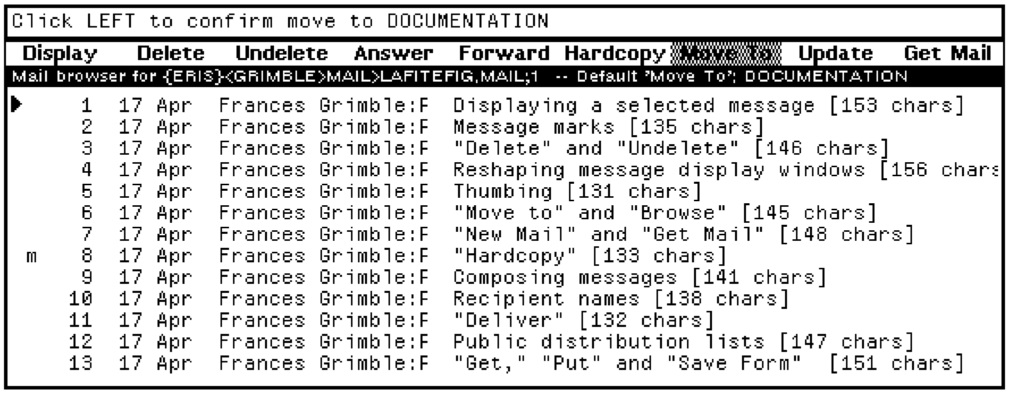
8. Filing Messages

Most users prefer not to keep all their messages in their one Active Mail folder, but rather disperse them into several different folders, usually sorted by topic. This makes it easier to locate old messages of interest. Lafite allows you to create additional mail folders as you wish and to move your messages freely among them.

The Move To Command

To move messages from one folder to another, select the message(s) that you wish to move. Then click on the Move To command with the left mouse button. A menu will appear listing your existing mail folders, plus the item ‘‘Other Folder.’’ If you wish to move the messages into one of the existing folders, simply select the folder’s name from the menu. Lafite then prompts you to confirm the move: ‘‘Click LEFT to confirm move to *FOLDER.*’’ If you click the left mouse button, Lafite completes the move, deleting the message(s) from the first folder’s table of contents and marking them with an *m.* If you click the right mouse button, the move is aborted.

If you wish to move the messages into a folder not listed on the menu or to a new folder, select the ‘‘Other Folder’’ item. Lafite will prompt you with ‘‘*FILENAME* (CR to abort):’’ in the prompt region. Type the name of the folder, followed by a carriage return (or a bare carriage return if you change your mind and want to abort the command). Lafite then prompts you as above to confirm the move: ‘‘Click LEFT to confirm move to *FOLDERNAME,*’’ followed by ‘‘[*OLD FILE*]’’ if the file you named already exists, or ‘‘[*NEW FILE*]’’ if it doesn’t (see figure 10). If you click the left mouse button, Lafite performs the move, creating a new folder if necessary, and adds the folder’s name to the menu for future selection.



```
Click LEFT to confirm move to DOCUMENTATION
 Display    Delete   Undelete   Answer   Forward  Hardcopy  Move To   Update   Get Mail
Mail browser for {ERIS}<GRIMBLE>MAIL>LAFITEFIG.MAIL;1  -- Default 'Move To': DOCUMENTATION
▶     1  17 Apr  Frances Grimble:F  Displaying a selected message [153 chars]
      2  17 Apr  Frances Grimble:F  Message marks [135 chars]
      3  17 Apr  Frances Grimble:F  "Delete" and "Undelete" [146 chars]
      4  17 Apr  Frances Grimble:F  Reshaping message display windows [156 chars
      5  17 Apr  Frances Grimble:F  Thumbing [131 chars]
      6  17 Apr  Frances Grimble:F  "Move to" and "Browse" [145 chars]
      7  17 Apr  Frances Grimble:F  "New Mail" and "Get Mail" [148 chars]
 m    8  17 Apr  Frances Grimble:F  "Hardcopy" [133 chars]
      9  17 Apr  Frances Grimble:F  Composing messages [141 chars]
     10  17 Apr  Frances Grimble:F  Recipient names [138 chars]
     11  17 Apr  Frances Grimble:F  "Deliver" [132 chars]
     12  17 Apr  Frances Grimble:F  Public distribution lists [147 chars]
     13  17 Apr  Frances Grimble:F  "Get," "Put" and "Save Form"  [151 chars]
```

Figure 10. A browser window showing a Move To confirmation prompt and the name of the current default Move To folder

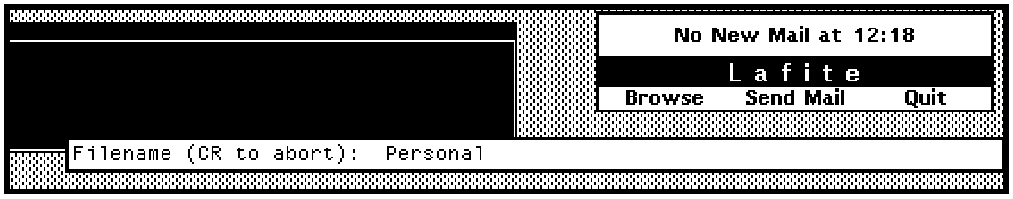
In either case, once the move is complete, the information ‘‘Default ’Move To’: *FOLDERNAME*’’ appears in the title bar of the browser.

This is for the benefit of the ‘‘accelerated’’ form of the Move To command. If you select the Move To command with the middle mouse button, Lafite performs the move to the folder named in the title bar without bringing up a menu of choices.

If you are already viewing the destination folder of a Move To in a browser on your screen, the newly moved messages are appended immediately to the browser’s table of contents (although the browser is not scrolled to show them, as it would be with the Get Mail command).

How to Create New Folders

A new folder is automatically created if you ask to move mail into a folder that doesn’t exist yet. You can also create a new folder without using the Move To command. Click on the Browse command from the Lafite status window and select ‘‘Other Folder’’ from the menu that appears. A small prompt window will appear with the prompt ‘‘FILENAME (CR to abort)’’ (see figure 11). Type the desired name of the folder and press the carriage return. If no folder of that name already exists, Lafite prompts you to confirm that you want a new folder, to make sure that you didn’t merely mistype the name of an existing folder: ‘‘Click LEFT to confirm creating *FOLDERNAME.*’’ If you click the left mouse button, Lafite will create the folder and prompt you to shape a browser window for the folder by showing a rectangle of dashed lines. The browser’s table of contents will initially be blank, and the prompt region will report ‘‘Folder is empty.’’ You can now retrieve new mail to this folder, using the Get Mail command, or move messages into it from another folder, as described above.



Filename (CR to abort): Personal

Figure 11. The prompt window for creating a new mail folder from the status window. The user first selected the Browse command, then the Other Folder item from the folder menu

How to View Other Folders

When you start up Lafite without specifying which folder you wish to view, Lafite automatically browses your default folder (usually ACTIVE.MAIL). If you instead wish to view a different mail folder at start-up time, then start Lafite by typing (LAFITE ’ON ’*FOLDERNAME*) to the executive window. Once Lafite is started, you can view any folder by clicking on the Browse command from the Lafite status window and selecting the folder from the menu offered, or selecting ‘‘Other Folder’’ and typing its name.

The Forget Folder Command

Once you create a folder, it remains in your folder menu (the one offered by the Browse and Move To commands) until you either expunge all messages from the folder, thereby deleting the folder, or you explicitly ask that the folder be removed from the menu. This latter way is needed, for example, if you have browsed various public mail folders that you no longer wish to see in your folder menu.

You can remove a folder from the Browse menu by invoking the Browse command with the middle mouse button. The resulting menu provides four options: Browse, Forget Folder, Forget Message Form, and Browse Laurel File (see figure 12). Browse Laurel File can be used only within Xerox. The Browse command is the same as the one in the Lafite status window, and the Forget Form command is discussed in chapter 9, ‘‘Writing Messages.’’ To use the Forget Folder command, simply click on it with the left mouse button and select the folder you want to remove with the same mouse button. The Interlisp-D prompt window will tell you that the folder has been forgotten.

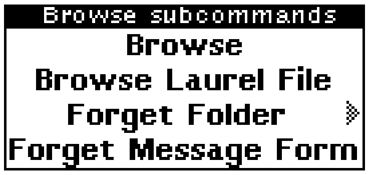


Figure 12. The middle-button Browse menu, showing the options for browsing files and deleting menu items

‘‘Forgetting’’ a folder only removes the folder from the menu; it does not delete the file. You can always browse the folder again by selecting Other Folder from the menu and typing the folder’s name. Within Lafite, you can permanently delete a folder by deleting all its messages, expunging it, and closing its browser window. In this case, the folder’s name is automatically removed from the folder menu. You can also delete a folder outside of Lafite by using the File Browser Lisp Library package to delete and expunge the folder and its table-of-contents file (named *FOLDERNAME*.MAIL-LAFITE-TOC), or using the function DELFILE to delete those two files, by typing (DELFILE *’{FILESERVER}<DIRECTORY>FOLDERNAME*) to the top-level executive. In either case, Lafite is unaware that you have deleted the folder behind its back, so you must use the Forget Folder command if you wish to remove the folder’s name from the folder menu.

9. Writing Messages

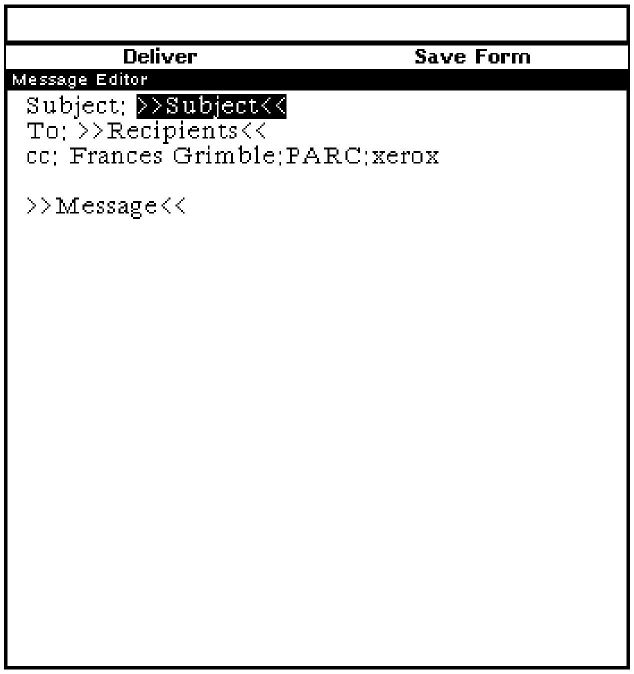
Writing a message is the process of composing its header and body. *Sending* a message is the process of transmitting the message to its specified recipients. This chapter gives instructions for writing and editing messages, and chapter 10 tells you how to send them.

Lafite provides five kinds of forms you can use for writing messages. You may *compose* a new message, or *answer* one you have received, or *forward* existing messages to a new recipient. You can also use specialized message forms to *report* problems or suggestions regarding Interlisp, Lafite, or TEdit, or use message forms that you have personally composed.

All the message forms have the same essential structure, but they have different kinds and amounts of information already filled in when presented to you for further editing. Some message forms are built by browser commands; others from the Send Mail command. The characteristics common to every type of message form are described first, then the differences between the forms. Once you select a message form, the process of writing and sending each kind of message is the same.

The Message Composition Window

Each type of form is contained in a message composition window (see figure 13). The message composition window contains four regions. The topmost region is the prompt region, where Lafite gives you messages and sometimes prompts you for information. Below it is a menu with two commands, Deliver and Save Form; these are explained in chapter 10. The black title bar informs you that you are in the message editor. At the bottom is the message composition region, which has two parts separated by a blank line: the header and the body.



```
Subject: >>Subject<<
To: >>Recipients<<
cc: Frances Grimble:PARC:xerox

>>Message<<
```

Figure 13. A message composition window showing a Standard message form. The top three lines of the text compose the skeleton message header; the text >>Message<< shows where you can begin the body of your message

The Message Header

The Standard form shown in figure 13 contains a *skeleton message header*. This header has two blank *fields* to be filled in that provide information to your recipient and enable Lafite to deliver your message. These are the Subject field and the To field. The third field, the CC field, is optional; Lafite automatically inserts your name in it.

The skeleton message header contains only the fields that you normally need to fill in yourself to send a message. There are a large number of fields that can possibly appear in a message header, as you can see from viewing the existing mail in your mail folders. Some fields (the date and sender information) are added automatically when Lafite delivers the message; some are added by mail servers along the way; and some are fields that you can add yourself if there is a need.

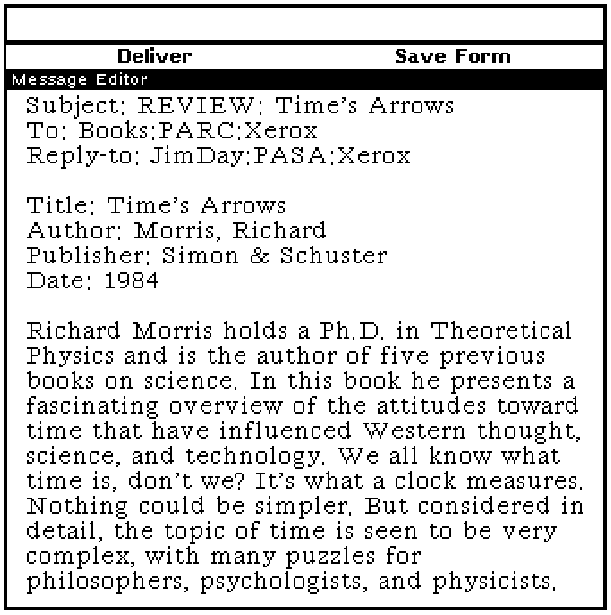
Each separate field of the header begins on a new line with a word immediately followed by a colon. The word preceding the colon specifies the meaning of the text following the colon. For example, To: specifies that the words following the colon are to be interpreted as recipient names, and Subject: specifies that the text following the colon is the subject of the message, which is of interest to the recipient but not part of the delivery information.

Lafite identifies items it expects you to replace by supplying keywords between reversed, double angle brackets. To make it easier to fill in these items, the first field is highlighted as a delete selection. Typing any text will cause the keyword and brackets to be deleted. You can then fill in the field with the text of your choice. Each successive field (except the CC field) can be reached by pressing the Next key on the 1108 or 1186 keyboard (or the middle blank key on the 1132 keyboard). The final item in most message forms is >>Message<<, where you can begin the body of your message. You can also select fields with the mouse.

Because Lafite allows you to edit any part of your outgoing messages, you must observe certain rules to keep them intelligible to Lafite. The header must contain at least one recipient, and the names of multiple recipients in one field must be separated by commas. Fields must be separated by single carriage returns, so that each begins on a different line. The header and body must be separated by a double carriage return, leaving a blank line between them when the message is displayed in the Lafite window. All of Lafite's initial message forms have a blank line following the header. Do not remove it or insert a blank line inside the header; otherwise information contained in the header will be misunderstood by Lafite.

The Subject Field

The Subject field is used to state the topic of your message (see figure 14). The topic should accurately express the content of your message so that interested people will take the time to read the message, but uninterested people can delete it without reading it. An accurate Subject field also facilitates differentiating the message later on from others in the same folder. For example, if your message contains ideas for improving Lafite, the topic might be "Suggestions for new printing features in Lafite," rather than "Lafite" or "Suggestions."



Subject: REVIEW: Time's Arrows
To: Books:PARC:Xerox
Reply-to: JimDay:PASA:Xerox

Title: Time's Arrows
Author: Morris, Richard
Publisher: Simon & Schuster
Date: 1984

Richard Morris holds a Ph.D. in Theoretical Physics and is the author of five previous books on science. In this book he presents a fascinating overview of the attitudes toward time that have influenced Western thought, science, and technology. We all know what time is, don't we? It's what a clock measures. Nothing could be simpler. But considered in detail, the topic of time is seen to be very complex, with many puzzles for philosophers, psychologists, and physicists.

Figure 14. A message composition window showing several filled-in user-specified fields. Note that, because the message is addressed to a public distribution list, the sender has included a Reply-To field

The To Field

The To field (and the CC field, see below) is where you specify who is to receive your message. Lafite allows you to specify multiple recipients in the To field, provided their names are separated by commas. Conventionally, your main recipients are listed in the To field and the CC field is used to copy the message to additional recipients.

You need to know the registered address of each individual or group to whom you want to send a message. You can often discover these addresses by looking at the headers of incoming messages.

Recipient Addresses. NS recipient addresses have three parts—a *name*, a *domain*, and an *organization*—separated by colons, in the form ‘‘Cheryl Jones:PARC:Xerox.’’ A name can represent either a person or a group. Every name belongs to a particular domain, which is usually a small division of an organization that shares common resources, such as file servers and printers. And every domain belongs to an organization, which is usually a single company or a major division within a company. Thus, Cheryl Jones is an employee at PARC, a division of the company Xerox.

Within a domain, a person or group can have several alternative names known as *aliases*. An alias can always be used in place of a person or a group’s real name. Thus, if Cheryl Jones’s alias is Jones, her mail can be addressed to Jones:PARC:Xerox. If you want to create an alias for your name, see your network administrator.

Lafite allows you to omit the domain and/or organization for recipients who are in your own domain and organization, just as you may omit an area code when telephoning your neighbor across the street. However, you must include the domain and organization for other recipients. If a recipient’s domain or organization is left out and is not the same as your own, Lafite will redisplay the message, and the Interlisp prompt window will tell you which recipient’s address needs clarification.

Your default domain and organization (the variables CH.DEFAULTDOMAIN and CH.DEFAULTORGANIZATION) are usually set in your initialization file, or are set automatically if your network has only one accessible domain. You can change them by typing (DC INIT) to bring up a DEdit window on your initialization file, resetting the variables, then typing (MAKEFILE ’{*FILESERVER*}<*DIRECTORY*>*INITFILENAME*) to save the new version of the file. For more information on DEdit and the MAKEFILE function, see the *Interlisp-D Reference Manual*.

Public Distribution Lists. The mail system provides a way to address messages to groups of recipients called *public distribution lists*. NS public distribution list addresses have the same form as individuals’ addresses. Using a distribution list as the recipient of a message causes the message to go to all the individuals included in the group. For example, the To line ‘‘To: AISBU:PARC:Xerox’’ will cause the message to be delivered to all the members of the Artificial Intelligence Systems Business Unit in the Palo Alto Research Center domain.

Public distribution lists are stored and maintained on a Clearinghouse by a list owner or a network administrator; individual members can add themselves to some lists. If you want to create, add your name to, or remove your name from a public distribution list, contact your list owner or administrator, or use the NS Maintain program (documented in the *Lisp Library Packages Manual*).

While you are permitted to address a message to any public distribution list, you should think very carefully about your choice of message and list so as not to bother recipients with messages they don’t care to read. Check with experienced users to see which lists should be used for which kinds of messages (see also appendix A, ‘‘Using Lafite Courteously’’). Also, add a Reply-To field if appropriate (see below).

You need not worry about a recipient appearing in more than one distribution list. The mail system will detect duplicate names among recipients and ensure that each receives the message only once. Thus, you may use multiple distribution lists freely within the To and CC fields of the message header.

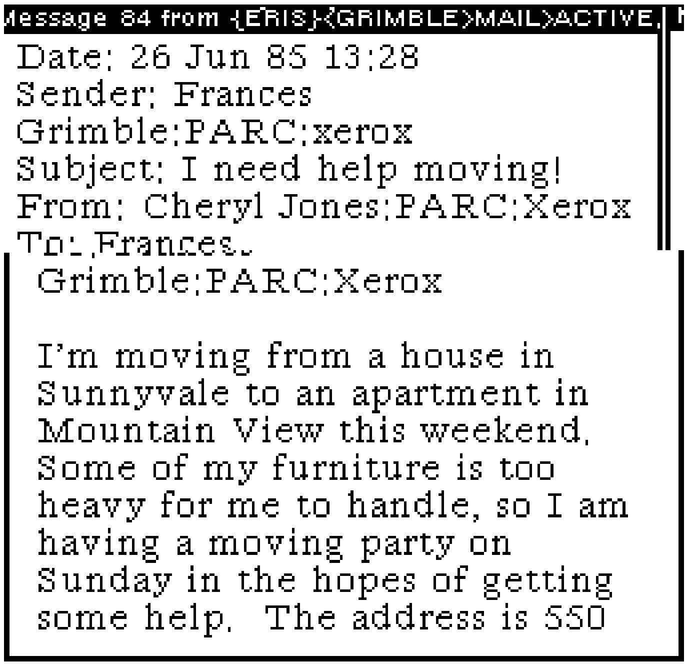
NS Maintain enables you to obtain lists of names, aliases, domains, organizations, and members of groups on your Clearinghouse network. For information, see the *Lisp Library Packages Manual*.

The CC Field

The CC field is used to list persons other than your main recipient(s) who are to receive your message. Names must be separated from each other by commas. When you send your message, these people will automatically receive it along with those specified in the To field. Lafite automatically puts your name in the CC field; you can delete it if you wish. If you wish to send no copies of the message, not even to yourself, delete the entire field.

The From and Sender Fields

Lafite automatically supplies a From field containing your name, which is not visible before the message is delivered. If you want the message to appear to be ‘‘from’’ someone else (e.g., if you are sending the message from somebody else’s logged-in Lafite), or from more than one user, you can supply your own From field. In this case, Lafite will supply a Sender field to show who actually sent the message (see figure 15).



```
Message 84 from {ERIS}<GRIMBLE>MAIL>ACTIVE.
Date: 26 Jun 85 13:28
Sender: Frances
Grimble:PARC:xerox
Subject: I need help moving!
From: Cheryl Jones:PARC:Xerox
To: Frances
Grimble:PARC:Xerox

I'm moving from a house in
Sunnyvale to an apartment in
Mountain View this weekend.
Some of my furniture is too
heavy for me to handle, so I am
having a moving party on
Sunday in the hopes of getting
some help. The address is 550
```

Figure 15. A message display window showing the From and Sender fields. In this case, the sender (Cheryl Jones) wrote and sent the message from the recipient’s logged-in Lafite

The Date Field

During delivery, Lafite inserts a Date field in the form ‘‘*Day Month Year Time* [in international hours].’’ Thus, if you send a message on April 1, 1985 at 1:45 p.m., the Date field will read ‘‘1 Apr 85 13:45.’’ You cannot supply your own Date field.

The Reply-To Field

When sending a message to a distribution list, or to a large number of individual users, it is important to keep in mind your audience and what you want to have happen if someone wishes to answer your message. Very often it is the case, especially with large distribution lists, that you want replies to be sent only to you, or to some small designated set of people. For example, you send a message announcing a meeting and want people to tell you if the time is bad for them. Or you are taking a poll. Or you are simply making an announcement to which you don't even expect replies. Message system etiquette demands that messages not be inappropriately sent to large numbers of people, as would ordinarily occur if a recipient of your message used the Answer command and didn't take the time to notice that a distribution list was copied on the resulting message.

It is for this reason that the Reply-To field exists. If a message contains a Reply-To field, most message-handling systems, including Lafite, will compose answering messages addressed only to the user(s) named in the Reply-To field, not to the entire recipient list of the original message. To add a Reply-To field to your message, simply insert a line beginning ‘‘Reply-to:’’ followed by one or more addresses (in the same form as the To field), anywhere in the header (but conventionally at the end of the header). Be sure you still have a blank line between the header and the body of the message.

The In-Reply-To Field

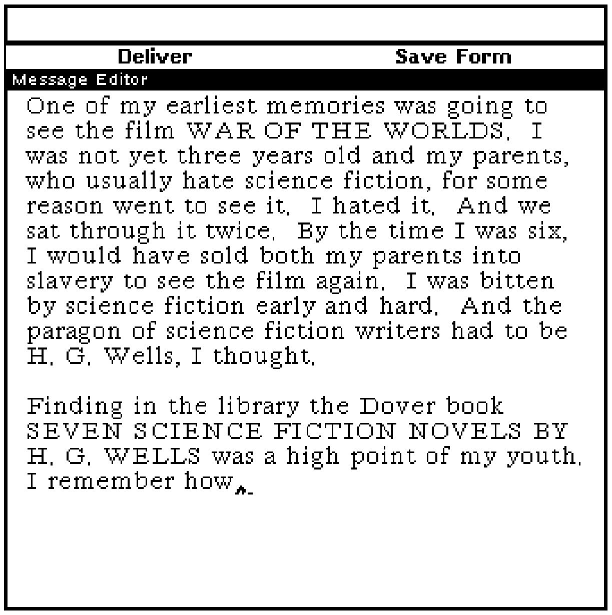
When you use the Answer command (see below) to reply to a message, Lafite automatically adds an In-Reply-To field to the message form. This field contains your name and the date of the message you are answering, in the form ‘‘In-reply-to: Cheryl Jones:PARC:Xerox's message of 5 Mar 85 17:29.’’ You can edit it if you wish.

The Message Editor

You have available to you the full power of TEdit when composing messages. Lafite message composition windows are TEdit windows and are used in the same way. Because TEdit has many more features than can be covered here, we will describe only the most basic procedures for writing and editing messages. If you are familiar with TEdit, you can skip this section. If, after reading this section, you want to know more about TEdit, see the TEdit documentation in the *Lisp Library Packages Manual*.

The Type-in Point

The blinking caret in the message composition window is called the *type-in point* (see figure 16). The Interlisp environment has only one type-in point active at a time; you tell the environment you want to type in the message composition window by clicking a mouse button on that window. Now whatever text you type will appear at the type-in point.



One of my earliest memories was going to see the film WAR OF THE WORLDS. I was not yet three years old and my parents, who usually hate science fiction, for some reason went to see it. I hated it. And we sat through it twice. By the time I was six, I would have sold both my parents into slavery to see the film again. I was bitten by science fiction early and hard. And the paragon of science fiction writers had to be H. G. Wells, I thought.

Finding in the library the Dover book SEVEN SCIENCE FICTION NOVELS BY H. G. WELLS was a high point of my youth. I remember how

Figure 16. A message composition window showing the type-in point and text wraparound. The caret after “how” is the type-in point. The text will be adjusted automatically when the window is reshaped or the message is printed

TEdit automatically breaks text between words and sends the overflow to the next line. Don’t use a carriage return when you think you have reached the end of a line; use one only when you want to insert blank space between paragraphs or sections. TEdit automatically adjusts the text of your message to fit the width of your recipient’s message display window or your printer’s requirements, but it leaves blank lines in place.

Text Selection

You may want to change the text of your message after you type it in by deleting it, copying it, or moving it. To change the text of your message, you first say where you want the change made by making a selection. Then you say what you want done by giving a command. Just making a selection has no effect on the message; only commands can change it. Each selection supersedes the previous one; that is, there is only one selection at a time. Deletion and other operations are applied to the currently selected text. You can tell when text is selected because it is highlighted by underlining or reverse video (usually white-on-black); it may have a caret flashing at one end.

There are two regions within an editor window, and which region the mouse cursor is in determines what kind of selection happens. The regions are the area containing text and the line bar that forms the left border of the window (this is not the same as the scroll bar described in chapter 5).

You select text by first pointing with the mouse, then pushing one of its buttons. The left mouse button always selects the smallest units. In the text region, it selects the character you’re pointing at; in the line bar, it selects the single line you’re pointing at.

The middle mouse button selects larger units. In the text region, it selects the word the cursor is over, and in the line bar it selects the paragraph the cursor is next to.

The right mouse button always extends a selection. If you select a place in the text with the left or center mouse button, move the mouse cursor somewhere else, and click the right mouse button, all the text in between the two points is selected. If the existing selection is a whole line or paragraph, the extended selection is also a whole line or paragraph.

Text Deletion

There are three ways you can select text for deletion. If you hold down the control key while selecting text, the selected text will be deleted when you release the key. You can also delete text by selecting it and then pressing the Delete key. You can delete text one character at a time by pressing the backspace key; the character to the left of the caret will disappear. Figure 13 shows some text highlighted as a delete selection.

Text Moving

To move text, first click the left mouse button to put the caret in the place you want the text moved to. Then select the text to be moved while holding down the Move key (or the control and shift keys on an 1132 workstation). When you release the key(s), the text will be moved to where the caret is. You can use this method to move text within a message composition window or to move it from one window to another.

Text Copying

If you want to copy text, first put the caret in the place you want the text copied to. Then hold the Copy key down while selecting the text to be copied. When you release the Copy key, the text will be copied to the location of the caret. You can copy text within a window or from one window to another. However, the most efficient way to copy an entire file from one window to another is to use the Include or the Get command; these are described below.

The Include Command

It is often desirable to electronically mail a document, such as a report, that is not a letter. You can conveniently do this by including that file in a Lafite message. First, you open a message composition window and fill in the header information. Next, you type in any covering message you want to send and place the blinking caret where you want to insert the file. Then put the mouse cursor on the black title bar of the window and press the left or middle mouse button.

The basic TEdit command menu will pop up; it contains the commands Put, Get, Include, Find, Looks, Substitute, Quit, and Expanded Menu (see figure 17). While you can use all these commands from within Lafite, only the Include, Put, and Get, commands are described in this manual. For detailed information you should consult the TEdit documentation.

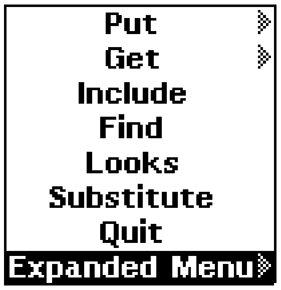


Figure 17. The basic TEdit command menu. Selecting Expanded Menu will bring up a menu containing most of the commands on the

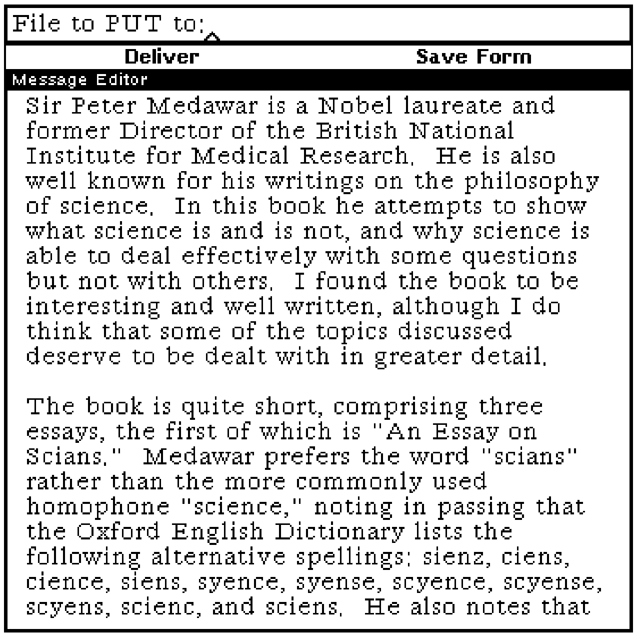
basic menu, plus commands that enable you to access additional menus—the Character Looks, Paragraph Looks, and Page Layout menus

To include a file in a message, select Include with the left mouse button. TEdit will ask you for the name of the file you want to load. Type the file name in the prompt region and press the carriage return. After a pause to read the file, your message composition window will fill up with the designated file. The entire file will be highlighted by underlining. To get rid of the underlining, simply click the left mouse button anywhere in the file.

Be careful to give TEdit the exact name of the file you want to include; do not change spacing or punctuation. If TEdit tells you that it can't find a file, check to see if you typed the name correctly.

The Put Command

You can save messages in TEdit files using the Put command and finish or deliver them later on. Bring up the TEdit command menu as described above and select Put with the left mouse button. TEdit will ask you for the name of the file to put to (see figure 18). The syntax for naming files stored on your disk is {DSK}*FILENAME*. For files stored on a file server, it is {*FILESERVER*}<*DIRECTORY*>*FILENAME*. You don't need to specify a file server or directory if you want TEdit to use your default file server and directory. Make sure that no blank spaces appear in your file name.



Sir Peter Medawar is a Nobel laureate and former Director of the British National Institute for Medical Research. He is also well known for his writings on the philosophy of science. In this book he attempts to show what science is and is not, and why science is able to deal effectively with some questions but not with others. I found the book to be interesting and well written, although I do think that some of the topics discussed deserve to be dealt with in greater detail.

The book is quite short, comprising three essays, the first of which is "An Essay on Scians." Medawar prefers the word "scians" rather than the more commonly used homophone "science," noting in passing that the Oxford English Dictionary lists the following alternative spellings: sienz, ciens, cience, siens, syence, syense, scyence, scyense, scyens, scienc, and sciens. He also notes that

Figure 18. A message composition window after the user has chosen the Put command. The message will be saved and retrieved as a TEdit file

Type the file name in the prompt region and press the carriage return. If you have already saved the file, TEdit will prompt you with the file name you specified earlier. You can either change this file name or confirm the earlier name by pressing the carriage return. TEdit will tell you that it is putting the file, then that it is done. You can now close your message composition window and later retrieve the message with the Get command.

The Put and Get commands are very useful for saving messages that can't be delivered due to network problems. (Or you can shrink the message composition window like any other Interlisp window; the icon looks like an envelope and is labeled "Unsent.")

You can also save messages with the Save Form command and retrieve them from the middle-button Send Mail menu; see below.

The Get Command

To retrieve a previously composed, undelivered message, bring up a Lafite message form and select Get from the TEdit menu with the left mouse button. TEdit will prompt you for the file to get. Type its name in the prompt region and press the carriage return. The mouse cursor will change to an hourglass until TEdit finishes getting the file. Once the file is loaded, you can edit or send your message. If you get a Lafite mail folder into TEdit and edit it, change the file's name when you save it. If you continue to use the file as a Lafite folder, Lafite will be unable to parse the folder the next time you open it (see chapter 12, ‘‘Troubleshooting Lafite Problems’’).

The Hardcopy Command

You can use the Hardcopy command in the Interlisp-D window menu to print messages before you send them. You bring up the window menu by placing the mouse cursor on the title bar of a message composition window and pressing the right mouse button. The menu of window manipulation commands shown in figure 19 will appear. Select Hardcopy and wait for TEdit to tell you it is done formatting the message for print. You may have to wait a little longer before your printer has it ready for you.

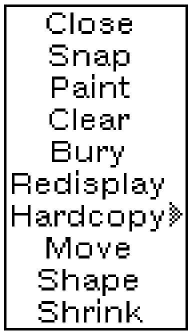


Figure 19. The Interlisp-D right-button window menu. The TEdit documentation in the Lisp Library Packages Manual *fully describes how to use this menu with the text editor*

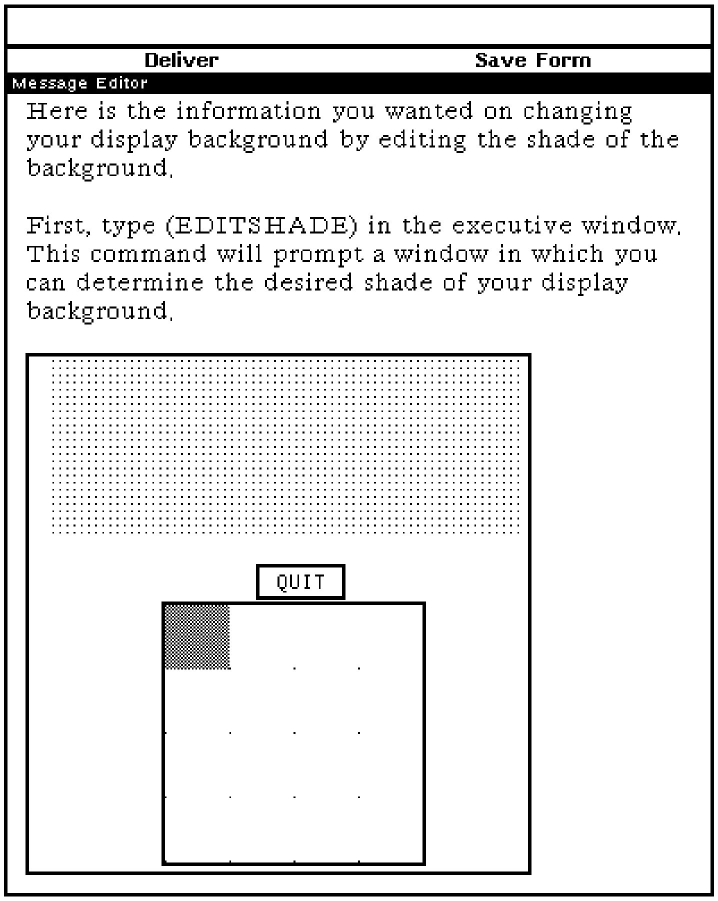
Fancy Message Formats

Lafite supplies you with an attractive default message design. If you don't specify otherwise, your messages are displayed in a 12-point Times Roman font. They are printed in a 12-point Classic font on an Interpress printer or a 12-point Times Roman font on a Press printer. They have no line or paragraph leading, and the margins are set flush left and ragged right.

If you wish, you can change this design using TEdit's font and formatting options. (You can also change Lafite's default fonts by resetting global variables; see chapter 13, ‘‘Customizing Lafite.’’) These include several additional fonts and type sizes, boldface and italic type, justified margins, and tabs. Detailed instructions on designing documents with TEdit are contained in the TEdit documentation. You can send specially designed messages either formatted (containing the special format) or unformatted (in the default design). See chapter 10 for instructions on sending formatted messages.

Messages With Bit Maps

Bit maps can be included in messages sent to other Lafite users (see figure 20). To insert a bit map in a Lafite message, first get the picture onto your screen. Place the caret in the place in the message composition window where you want the image to appear. Hold down the shift key and depress the right mouse button in the background. This will bring up a rectangle for Snap, a window menu command that makes a copy of any portion of your screen and allows it to be placed elsewhere on the screen, for instance in a message composition window. Move the cursor into the rectangle, turning it white-on-black. When you release the key and button, the cursor changes from an arrow to a prompt design. Move the prompt (as if it were the cursor) to the bit map you want to include in the text. Press down the left button and outline the bit map as closely as possible, in the same way you would create a region for a window. When you let up the left button, the bit map will appear where you left the caret flashing in the message composition window.



Here is the information you wanted on changing your display background by editing the shade of the background.

First, type (EDITSHADE) in the executive window. This command will prompt a window in which you can determine the desired shade of your display background.

Figure 20. A Lafite message containing a bit map. It is often convenient to include a screen image in a Lafite message when you are giving technical information or reporting problems with the system

You can edit a bit map by clicking inside it with the left mouse button, which brings up a menu of editing operations. You can delete, copy, or move a bit map as if it were a single text character. If you insert a carriage return before and after a bit map, you can adjust its position with the TEdit paragraph-formatting menu (see the TEdit documentation). Bit maps are saved when you save a file, and they are printed when you hard-copy the message (providing your printer can print bit maps).

Messages containing bit maps are automatically sent as formatted messages; see chapter 10.

Message Forms

As described above, Lafite provides five kinds of forms you can use for writing messages, each of which is accessed by a different command or commands. The kinds of forms are the Standard form, the *Answer* form, the *Forward* form, the *Report* form, and the *Personally Created* (or saved) form. This section describes each form and how to use it.

The Standard Form

The Standard message form is a message composition window containing blank To and Subject fields and a CC field with your name in it (see figure 13). You can access the Standard message form in two ways. The first is to use the Send Mail command in the Lafite status window. If you click on this command with the left mouse button, Lafite will open a window containing the standard message form. If you click on Send Mail with the middle mouse button, Lafite will present you with a menu of several forms, including the names of any you have created yourself plus Lisp Report, TEdit Report, Lafite Report, Saved Form, and Standard Form (see figure 21). To access the Standard form, select it from the proffered menu. (Personally Created forms and Report forms are discussed below.)

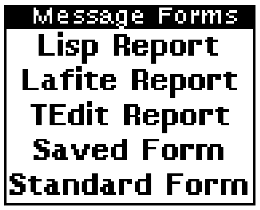
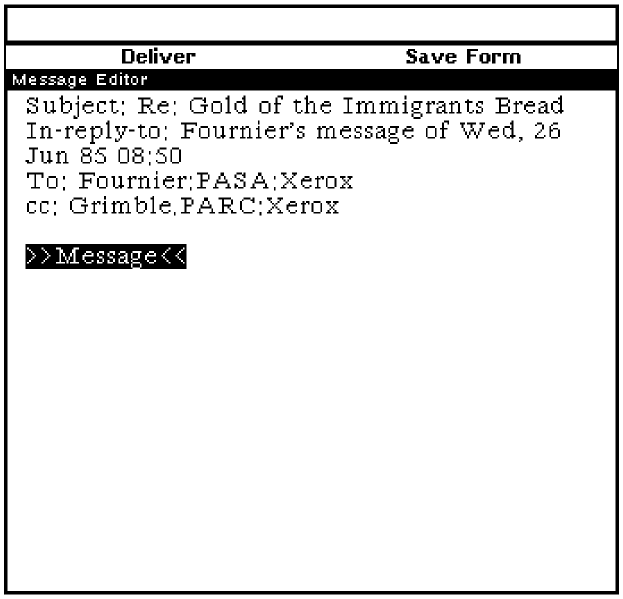


Figure 21. The middle-button Send Mail menu. In this case the user has no Personally Created forms

You can also access this second menu without otherwise entering Lafite by clicking on your screen background menu with the right mouse button and selecting SendMail from the menu that pops up. If you choose SendMail without having entered Lafite from the executive window, you cannot read, move, or otherwise manipulate messages—only write and send them using the Standard, Personally Created, and Report forms.

The Answer Form

Selecting the Answer command from the browser window menu constructs a message form that is a skeleton reply to the selected message or to the first selected message if more than one is selected (see figure 22). The header of the form contains To and CC fields listing the sender of the message and its recipients, a Subject field of the form ‘‘Re: *Subject of selected message*,’’ and an In-Reply-To field containing the name of the sender and the date and time the message was sent. For example, if you are replying to a message sent to you by Cheryl Jones on March 19, 1985 at 10:28 a.m., the field will be ‘‘In-Reply-To: Cheryl Jones:*Domain:Organization’s* message of 19 Mar 85 10:28.’’



Subject: Re: Gold of the Immigrants Bread
In-reply-to: Fournier's message of Wed, 26 Jun 85 08:50
To: Fournier:PASA:Xerox
cc: Grimble.PARC:Xerox

>>Message<<

Figure 22. An Answer form, showing the skeleton message header created by Lafite. The user can edit the header as he or she desires

The form provided by the Answer command is not always exactly what you might want. Lafite takes the viewpoint that it is easier to delete than to add text, and therefore includes all information that seems to be relevant. For example, Lafite often includes your name in the CC field of the answer form, even though you may not want a copy of the message you are composing. You should not expect that the answer form will be exactly right; always examine the header to be certain it contains the desired information.

In particular, if you reply to a message that was directed to a distribution list, the Answer command will copy the distribution list name into the CC field of the Answer form (unless the sender included a Reply-To field). You should consider carefully whether it is appropriate for your reply to be sent to that distribution list, and if not delete the distribution list's name.

After you deliver the answer to a message, an *a* will appear in the browser window as that message's mark.

The Forward Form

Selecting the Forward command from the browser window menu constructs a message form containing the complete text of all the selected messages, and a message header whose subject is the sender and subject of the first selected message, in brackets. You should fill in the desired recipients and a covering message if you desire one. You can also change the subject if you wish.

After you deliver a forwarded message, an *f* will appear as that message's mark.

Specialized Report Forms

The Report forms are used to report problems or make suggestions regarding Interlisp or an application program. Although they always appear on the middle-button Send Mail menu, you cannot use them unless your organization has set up a local support address or addresses to which you can send the reports (see chapter 13, ‘‘Customizing Lafite,’’ for instructions). If you choose one of the Report forms and an address has not been set up, a message will be printed in the prompt window saying that the address is not available. If you cannot use the Report forms, please move on to the section below on creating customized message forms.

The following three sections describe the headers of the Report forms, including the information that is provided automatically and the fields you fill in. We have assigned meanings to the fields and made recommendations for reporting problems and requesting features. These guidelines are included because we have found them to be effective and feel they may be helpful to our users. However, they may differ from the policies of your local support organization, so check with that organization before using these forms.

The Lisp Report

The Lisp Report has a header similar to that of the Standard form, plus a section giving certain information about the hardware and software you are using and asking you to supply information about your problem or feature request. The Subject field is filled in with ‘‘Lisp: >>Terse summary of problem<<.’’ Replace the words in reverse angle brackets with a subject that accurately identifies your problem or request. Subjects like ‘‘Floppy problem’’ are imprecise; instead, use subjects like ‘‘Attempt to write file when floppy door is open causes awful noise.’’ Feature requests generally start with the word ‘‘Want,’’ e.g., ‘‘Want to make windows circular rather than rectangular.’’

The To field contains the address of your local Lisp support. You do not need to modify it. The CC field initially contains your name.

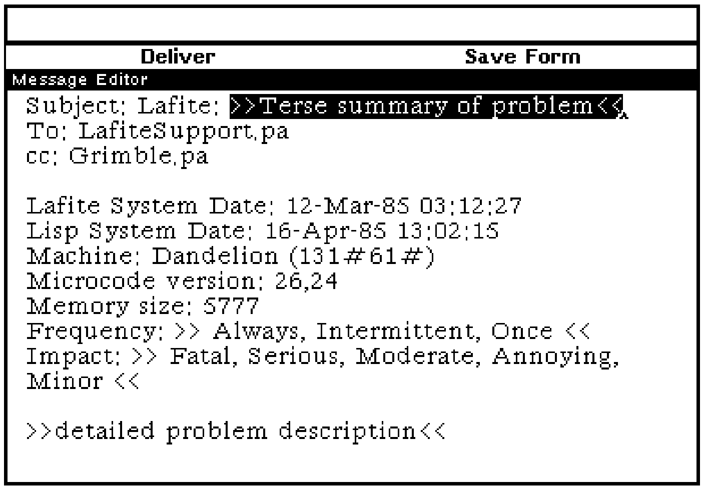
Lafite automatically fills in the date your sysout was made, the kind of workstation you are using, the microcode version, and the memory size. In addition, there is a Frequency field containing the words ‘‘>>Always, Intermittent, Once<<.’’ The Frequency field is used to report the reproducibility of a problem; it is usually irrelevant for feature requests. If the problem occurs every time you try to perform a task, choose Always and delete the angle brackets and the Intermittent and Once options. (This is most easily done by copy-selecting Always while the bracketed field is delete-selected.) If the problem doesn’t always happen, choose Intermittent; if you saw it happen once, choose Once. If the Frequency field is irrelevant, leave it blank or delete the whole line.

Use the Impact field to describe how seriously a problem or the lack of a feature affects your ability to get work done. The names apply to bug reports, but feature requests should be rated along analogous lines. Choose Fatal if the problem causes system crashes, loss of work, etc., or if a feature is required for project completion. Choose Serious if the problem can be worked around but seriously interferes with your work. Choose Moderate if the problem is tolerable but clearly a problem. Choose Annoying for annoying problems and requests for features that ‘‘would be nice.’’ Choose Minor for very minor feature requests or problems that may not even be bugs.

Replace the words ‘‘detailed problem description’’ with a description of the problem and what you were doing when you encountered it, plus any other information that might be useful. Always be as precise and detailed as possible; it is extremely difficult to solve a technical problem described only as ‘‘Floppy doesn’t work.’’

The Lafite Report

The Lafite Report differs from the Lisp Report in that the Subject field is filled out as ‘‘Subject: Lafite: >>Terse summary of problem<<’’ and the Lafite system date appears above the Lisp system date (see figure 23). You should report problems and requests for new features in the same way as in a Lisp Report.



```
Subject: Lafite: >>Terse summary of problem<<
To: LafiteSupport.pa
cc: Grimble.pa

Lafite System Date: 12-Mar-85 03:12:27
Lisp System Date: 16-Apr-85 13:02:15
Machine: Dandelion (131#61#)
Microcode version: 26,24
Memory size: 5777
Frequency: >> Always, Intermittent, Once <<
Impact: >> Fatal, Serious, Moderate, Annoying,
Minor <<

>>detailed problem description<<
```

Figure 23. The Lafite Report form, showing the fields you need to fill in and the ones Lafite fills in automatically

The TEdit Report

The TEdit Report is exactly like the Lafite Report, except that ‘‘TEdit’’ appears in the subject instead of ‘‘Lafite,’’ and the TEdit system date is given instead of the Lafite system date.

Personal Message Forms

You can create, save, and reuse Personal message forms or form letters with Lafite’s Save Form command. Simply write your message in the message composition window, then choose Save Form from the window’s menu with the left mouse button. Lafite will print the following message in the prompt region: ‘‘Save form under name:.’’ Type the name you want the form to have (it cannot contain a space) and press the carriage return. If no extension is given for the file name, Lafite defaults it to Lafite-Form. Lafite will print the full file name in the prompt region, then close the message composition window.

When composing your Personal form, you will probably want to use the same convention as the regular Lafite forms do for items that will need to be filled in when the form is used to send a message. That is, enclose a describing word or phrase in reverse angle brackets, so that TEdit’s Next key can be used to easily skip from one such item to the next. In addition, if you include the text ‘‘>>Self<<’’ anywhere in the message, it will be automatically replaced with the name of the logged-in user when the form is requested. The most common use of this is to put the line CC: >>Self<< in the header. This is better than putting your own name there, since your form can be used by other people without having to edit the CC field. Of course, if you edit such a form and save it away again, you must remember to replace your name with >>Self<< again, since the >>Self<< is replaced with your name when you bring up the form.

There are two ways you can access Personally Created forms. After you save a form, its name is added to the middle-button Send Mail menu, and you can select it just as you would any of the other forms on that menu. You can also select Saved Form from the same menu; this brings up a small window asking for the file name. Type in the name of the form and press the carriage return.

When you select, edit, and deliver a Personally Created form, you are only editing a copy of the form; the original remains unchanged. Of course, if you want to change the actual form you can edit it and save it again with the Save Form command.

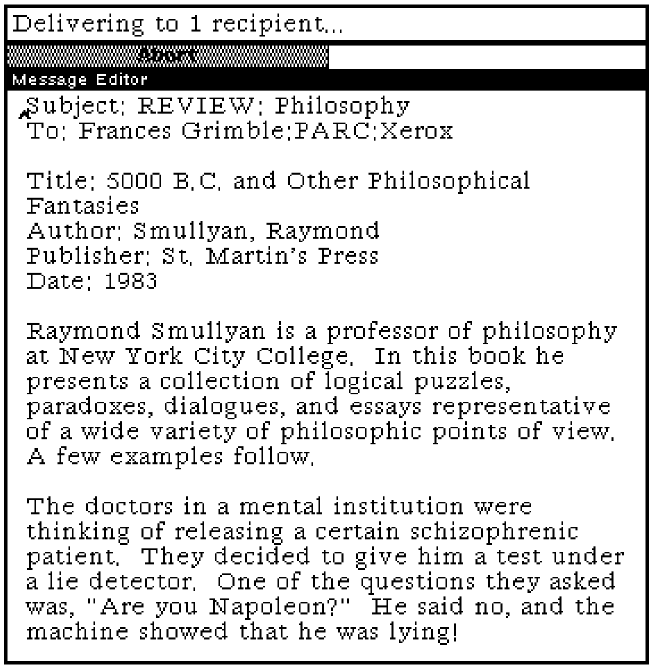
Note that Save Form is intended for the creation of new message *forms*. If you merely wish to save a partially composed message for later completion, TEdit's Put command is more appropriate.

The Forget Message Form Command

Once you have saved a form, its name will continue to appear in the middle-button Send Mail menu. If you no longer want that form in the menu, use the Forget Message Form command in the middle-button Browse menu to remove it. Select Browse with the middle mouse button, then Forget Message Form from the proffered menu (see figure 12). This brings up a menu of Personally Created forms; select the one you want to forget with the left mouse button. The Interlisp-D prompt window will tell you the form has been forgotten. Note that, as with the Forget Folder command, this does not delete the actual file. To delete the file, type (DELFILE '{*FILESERVER*}<*DIRECTORY*>*FILENAME*.LAFITE-FORM) in the executive window. You can also use the File Browser Lisp Library package.

10. Sending Messages

Once you have written the message you want to send, you can initiate its delivery to the recipient by pointing the cursor at Deliver and clicking the left mouse button. Lafite will fill in your name and the date (though they won't appear in the window) and proceed to send the message. A gray background will appear behind the Deliver command, then the command will change to Abort (see figure 24). If you decide not to send a message after buttoning Deliver, select the Abort command and Lafite will halt the delivery of the message.



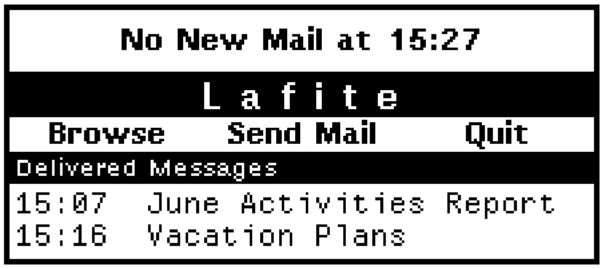
```
Delivering to 1 recipient...

Subject: REVIEW: Philosophy
To: Frances Grimble:PARC:Xerox

Title: 5000 B.C. and Other Philosophical
Fantasies
Author: Smullyan, Raymond
Publisher: St. Martin's Press
Date: 1983

Raymond Smullyan is a professor of philosophy
at New York City College.  In this book he
presents a collection of logical puzzles,
paradoxes, dialogues, and essays representative
of a wide variety of philosophic points of view.
A few examples follow.

The doctors in a mental institution were
thinking of releasing a certain schizophrenic
patient.  They decided to give him a test under
a lie detector.  One of the questions they asked
was, "Are you Napoleon?"  He said no, and the
machine showed that he was lying!
```

Figure 24. A message composition window after Abort was chosen. After Lafite aborts the delivery, the word "abort" will appear in the prompt region and the message will be redisplayed

Invoking Deliver will send the message to all its intended recipients. Each recipient will receive only one copy of the message, even if his or her name appears more than once (a name might be on several distribution lists specified in the To or CC field).

If Lafite discovers an error in the list of recipients, it will tell you that the "recipient is not understood" and give you an opportunity to correct the mistake and invoke Deliver again. When the list is acceptable (i.e., all specified recipients are known to have in-boxes), Lafite will deliver the message. After successful delivery, the word Abort will disappear. Lafite will tell you that it is done and close the message composition window. Delivery happens in the background, so you can perform other tasks while delivery proceeds.

After a message is delivered, it is entered into your out-box, a window attached to the bottom of your status window (see figure 25). This window contains one-line descriptions of the two messages you sent most recently. The out-box is treated as a menu—selecting a line in it brings up the corresponding message for further editing and delivery. The out-box can be independently closed if you are no longer interested in the messages displayed therein.


No New Mail at 15:27

Lafite

```
15:07  June Activities Report
15:16  Vacation Plans
```

Figure 25. A status window showing the out-box. The number of messages retained can be changed using the LAFITEOUTBOXSIZE variable; see chapter 13, ‘‘Customizing Lafite’’

If you decide not to do anything with a message you started to compose, simply close the message composition window with the standard right-button window menu. Lafite asks you to confirm flushing the message; its text will not be saved.

Delivery of Formatted Messages

Only Lafite users can read a formatted message; all other mail readers will see only the plain text of the message. If you try to deliver a formatted message, Lafite asks if you want to retain the formatting information, putting up a menu with the choices Send Formatted Message, Send Plain Text, and Abort. Send Formatted Message retains all the formatting information. This choice is made automatically if the message contains a bit map, because there is no way to send the image without the formatting. If you send a message containing a bit map to a non-Lafite user, the content of the bit map will appear as a string of trash, which may disturb some message-reading systems.

Send Plain Text sends only the text of the message. The message will appear to the recipient in whatever font his or her mail reader usually uses, and all paragraph formatting (centering, justification, special tab stops) will vanish. Sometimes a message appears to Lafite to be ‘‘formatted‘‘ when it really only contains inconsequential variations in fonts or line spacing, typically the result of copy-selecting text from another window. In such cases, where you did not intend special formatting, it is best to choose Send Plain Text to reduce the message overhead and avoid imposing your default font choice on your recipients.

Abort does not send the message, but returns you to the message editor to continue editing the message.

11. Leaving Lafite

There are two ways you can leave Lafite. You can select Quit from the status window (or the middle-button Quit menu) which stops Lafite and closes all browser windows. If any of your open mail folders need updating, Lafite prompts you with a menu asking what degree of updating should be performed: Write Out Changes Only, Expunge Deleted Messages, Don't Update File, and Don't Quit; the option you select is applied to all open folders (see figure 26). You must select one of the first three options before Lafite will let you quit. When you do, all your Lafite windows will disappear, indicating that you have successfully quit Lafite.

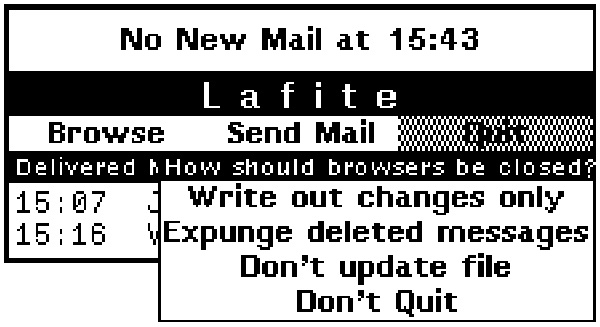


Figure 26. The update menu that appears when you choose Quit and an open mail folder (or folders) needs updating

You can also leave Lafite by typing (LAFITE 'OFF) in the executive window, which has the same effect as selecting Quit.

If you select Quit with the middle mouse button, you get a menu that includes the original Quit command and the Restart command. The Restart command turns Lafite off and then immediately back on again—it is equivalent to typing (LAFITE 'OFF) followed by (LAFITE 'ON NIL). It is mainly useful if you have changed some of your Lafite personalization and you want Lafite to notice the new settings.

You may want to keep the Lafite status window in view instead of quitting Lafite each time you finish reading or sending a batch of mail. The status window keeps you constantly informed about the state of your in-boxes, and it takes up little room on your workstation screen.

[This page intentionally left blank]

12. TROUBLESHOOTING LAFITE PROBLEMS

Although Lafite is a robust system, you may occasionally encounter minor problems when using it. This chapter describes the most common Lafite problems and how to prevent and/or recover from them.

Lafite Confuses You With a Previous User

If you log into Lafite with the correct name and password but the Browse menu shows no folders or shows someone else's folders, Lafite may be confusing you with a previous user. If you are running in a sysout in which someone else has been using Lafite, you need to take some action to get Lafite to work on your mail files and in your name. When you change the user identity by logging in as yourself, Lafite notices that the current user has changed and attempts to authenticate you. However, Lafite still doesn't know how you want your Lafite customized; in particular, what your known mail folders are.

To get Lafite to recognize your identity, you should first turn Lafite off by choosing Quit from the status window or by calling (LAFITE 'OFF). Then log in again. Type (GREET) in the executive window; you will be asked for the name of your initialization file if there isn't one on your workstation's disk. When this has been loaded, restart Lafite.

File Server Is Slow

If your mail files are stored on a remote file server that is particularly unresponsive, the mail server connection over which new mail is retrieved may *time out* (that is, end communication with your file server) before the file server acknowledges receipt of the messages. The usual consequence of this is that your in-box is not flushed, so your new mail is in two places: your in-box, awaiting retrieval, and your mail file, to which it was just retrieved. A less common occurrence is that the mail server times out partway through the retrieval process, resulting in a Lisp break. You can type ^ after the prompt in the break window to return to the state before the Get Mail started.

If this is often a problem for you, you may want to adopt the following procedure to maintain the flexibility of remote mail files while utilizing the speed and reliability of the local disk. Keep most of your mail files on the remote server, as usual, but keep your Active Mail file, the one to which you usually retrieve mail, on your local disk. Retrieve mail to this file and dispatch from there to your remote files (using Move To) some or all of the messages you wish to keep. Mail files on disk have very predictable performance during Get Mail, which is good for both you and the mail server. Files on disk are also less subject to other vagaries of remote servers (e.g., sudden crashes) that sometimes cause problems with mail files (see figure 27). And if you tend to delete much of your incoming mail after reading it once, you may find it faster to keep your Active Mail on disk, even if your remote server isn't unreliable.

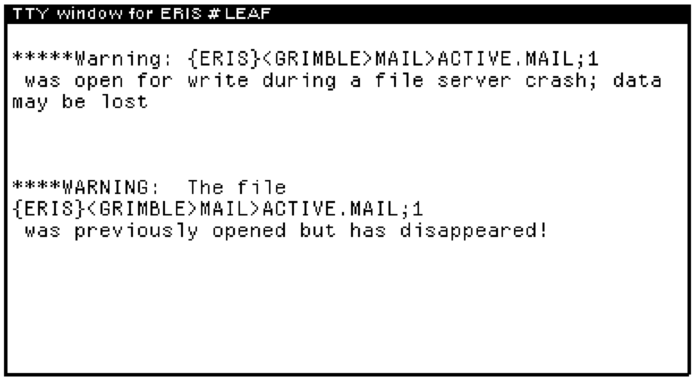
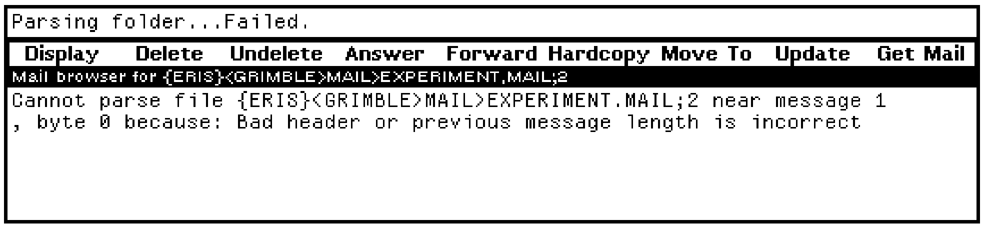

```
TTY window for ERIS # LEAF

*****Warning: {ERIS}<GRIMBLE>MAIL>ACTIVE.MAIL;1
 was open for write during a file server crash; data
may be lost



****WARNING:  The file
{ERIS}<GRIMBLE>MAIL>ACTIVE.MAIL;1
 was previously opened but has disappeared!
```

Figure 27. A warning window opened by Lafite after a file server crash. The folder has ''disappeared'' because the file server is down, therefore the mail folders on it cannot be accessed by Lafite

You can also choose to keep most or all of your mail files on disk, backing them up to a file server periodically. The Lisp Library package CopyFiles is helpful for doing the backup automatically.

Mail File Does Not Parse

The first time Lafite browses a mail folder, or any time that it suspects that the folder's table of contents is obsolete, it tries to *parse* the folder. A good mail folder consists of a set of messages, each set off by some internal information about the message's length and status. Parsing a folder consists of scanning for this internal information and building a table of contents from it. If a folder has been damaged in some way, the parsing operation fails, most commonly because a message's stated length is inconsistent with what is actually stored in the file (see figure 28). In this case, Lafite aborts the Browse command and prints a message in the browser window describing how far it got. The information Lafite prints includes the header of the last message parsed and a byte pointer (character position) in the file where the problem was encountered. The mail file is valid up to that pointer position.


```
Parsing folder...Failed.
```
Display Delete Undelete Answer Forward Hardcopy Move To Update Get Mail
Mail browser for {ERIS}<GRIMBLE>MAIL>EXPERIMENT.MAIL;2

```
Cannot parse file {ERIS}<GRIMBLE>MAIL>EXPERIMENT.MAIL;2 near message 1
, byte 0 because: Bad header or previous message length is incorrect
```

Figure 28. A browser window showing an error message printed by Lafite when it was unable to parse the mail folder

The most common way to get a mail file into an inconsistent state is to abort a Move To or Update command somewhere in the middle (either manually, or because a remote server crashed). In the case of Move To, the problem is usually that the last message in the file is ''too short,'' since it was never completely written. If the first operation you perform on the destination file after the crash is to browse it, Lafite will (usually) detect this situation and let you browse the file anyway, with a warning that the last message is truncated. Updating the file then corrects the length of the last message. If you neglect to browse the destination file before moving additional messages to it, however, your mail file will not parse.

Another way you can damage a mail file is to get it into a TEdit file, edit it, and save it under the original name. The next time you browse that mail folder, Lafite will be unable to parse the file.

The solution to ‘‘mail file does not parse’’ problems is to scavenge the mail file with the Lisp Library package Mail Scavenger, following the instructions in chapter 14, ‘‘Using Lafite-Related Lisp Library Packages.’’

Table of Contents Is Inconsistent With Mail File

Lafite breaks with a message to this effect when it tries to operate on a message that isn’t where Lafite thought it was in the file. The most common cause of this is aborting the Expunge command. The appropriate action is to close the browser, select Don’t Update, delete the table-of-contents file (the file with -LAFITE-TOC appended to the name) from your directory, and then browse the file again. If this is not successful, you may need to scavenge the file with Mail Scavenger. If you made many changes to the browser (deletions, for example) that you would rather not lose, you can try selecting Write Out Changes Only when you close the browser. This may succeed if the inconsistencies in the table of contents did not intersect with your changes.

13. CUSTOMIZING LAFITE

Some aspects of Lafite's behavior, such as whether each message of a batch you print starts on a new page and how many messages are shown in your out-box, are controlled by *global variables* set in your intialization file. Each variable is set to a *default value* that most users have found acceptable. However, you may want to reset some global variables to tailor Lafite to your needs. You edit Lafite variables in the same way as any other part of your intialization file, by typing (DC INIT) to bring up a DEdit window on your initialization file, resetting the variables, then typing (MAKEFILE *'{FILESERVER} <DIRECTORY>INITFILENAME)* to save the new version of your initialization file. For detailed information on DEdit and the MAKEFILE function, see the *Interlisp-D Reference Manual.*

Lafite's Global Variables

LAFITEDEFAULTHOST&DIR [Variable]

is the directory on which Lafite looks for the LAFITE.PROFILE and all mail folders and message forms you access if you don't supply an explicit directory. Lafite uses its own host and directory names for mail folders, etc., rather than the current connected directory because you may want to keep your mail folders someplace special (e.g., the local disk or your log-in directory), and the connected directory can change. LAFITEDEFAULTHOST&DIR is provided to tell Lafite where you usually keep your mail. LAFITEDEFAULTHOST&DIR should be in the same form as the variable LOGINHOST/DIR (i.e., *{FILESERVER}<DIRECTORY>)*. If LAFITEDEFAULTHOST&DIR is NIL, then the value of LOGINHOST/DIR is usedži.e., your log-in host and directory.

LAFITEFORMDIRECTORIES [Variable]

is a search path for Lafite forms, initially NIL. When you choose the Saved Form command underneath Send Mail, the form name that you enter is first searched for on your default directory (LAFITEDEFAULTHOST&DIR), and if it is not found there, Lafite searches the directories in the list LAFITEFORMDIRECTORIES. LAFITEFORMDIRECTORIES is typically set to a list of one or more public directories on which generally useful forms have been collected.

LAFITESTATUSWINDOWPOSITION [Variable]

specifies where the status window appears when Lafite is invoked. It is a POSITION or NIL (in which case you are asked to specify a position when Lafite starts).

LAFITEBROWSERREGION [Variable]

is a REGION used to describe where the primary (i.e., first) browser window is to be placed on the screen. Initially, it is set to something ''reasonable'' for the standard initial display configuration. If you set it to NIL, you are prompted to specify a region the first time any such window is created.

LAFITEDISPLAYREGION [Variable]

is a REGION used to describe where the primary (i.e., first) message display window is to be placed on the screen. Initially, it is set to something ‘‘reasonable’’ for the standard initial display configuration. If you set it to NIL, you are prompted to specify a region the first time any such window is created.

LAFITEEDITORREGION [Variable]

is a REGION used to describe where the primary (i.e., first) message composition window is to be placed on the screen. Initially, it is set to something ‘‘reasonable’’ for the standard initial display configuration. If you set it to NIL, you are prompted to specify a region the first time any such window is created.

DEFAULTMAILFOLDERNAME’s [Variable]

value is used if no mail folder is supplied to the function LAFITE, i.e., you call (LAFITE ’ON). Initially, this value is ACTIVE.MAIL.

LAFITEMAIL.EXT [Variable]

is the default extension for names of mail folders. It is initially MAIL.

LAFITETOC.EXT [Variable]

is the string appended to the name of a mail folder to produce the name of its table-of-contents file. It is initially LAFITE-TOC.

LAFITEFORM.EXT [Variable]

is the default extension for the names of user-defined form files. It is initially LAFITE-FORM.

LISPSUPPORT [Variable]

specifies the address to which Lisp Report messages are sent. Its value is a single string containing one or more addresses, all in the form you would put in the To line of a message header. If LISPSUPPORT is NIL, then the Lisp Report form cannot be used. It is initially NIL. Site administrators may want to set this variable in their site initialization files to the mail address of their local Lisp Support.

LAFITESUPPORT [Variable]

specifies the address to which Lafite Report messages are sent. Its value is a single string containing one or more addresses, all in the form you would put in the To line of a message header. If LAFITESUPPORT is NIL, then the Lafite Report form cannot be used. It is initially NIL. Site administrators may want to set this variable in their site initialization files to the mail address of their local Lafite Support.

TEDITSUPPORT [Variable]

specifies the address to which TEdit Report messages are sent. Its value is a single string containing one or more addresses, all in the form you would put in the To line of a message header. If TEDITSUPPORT is NIL, then the TEdit Report form cannot be used. It is initially NIL. Site administrators may want to set this variable in their site initialization files to the mail address of their local TEdit Support.

LAFITEDISPLAYFONT [Variable]

is a font used for displaying messages. You may change it without changing the other Lafite fonts. It should be a FONTDESCRIPTOR as returned by FONTCREATE. In your initialization file, this is usually specified with a FONTCREATE expression, e.g., (VARS (LAFITEDISPLAYFONT (FONTCREATE ’MODERN12))). Initially, it is Times Roman 12.

LAFITEEDITORFONT [Variable]

is a font used for composing messages. You may change it without changing the other Lafite fonts. It should be a FONTDESCRIPTOR as returned by FONTCREATE. Initially, it is Times Roman 12.

LAFITEHARDCOPYFONT [Variable]

is a font used for printing messages. You may change it without changing the other Lafite fonts. It should be a FONTDESCRIPTOR as returned by FONTCREATE. Initially, it is Times Roman 12, which prints as Classic 12 on an Interpress printer.

LAFITEBROWSERFONT [Variable]

is the font used for displaying the table of contents in the browser window. It is initially Gacha 10.

LAFITEMENUFONT [Variable]

is the font used for the items in all Lafite menus. It is initially Helvetica 10 Bold.

LAFITETITLEFONT [Variable]

is the font used for the title bar (‘‘Lafite’’) in the Lafite status window. It is initially Helvetica 12 Bold.

LAFITEENDOFMESSAGEFONT [Variable]

is the font in which the LAFITEENDOFMESSAGESTR is displayed (see below). It is initially Times Roman 10 Italic.

LAFITEENDOFMESSAGESTR [Variable]

is a string containing the text of the ‘‘End of Message’’ token displayed at the end of a message. If LAFITEENDOFMESSAGESTR is NIL, then no ‘‘End of Message’’ token appears.

LAFITENEWPAGEFLG [Variable]

is initially T. If so, the Hardcopy command starts each message on a new page. Otherwise it separates each message by a line of dashes.

LAFITEHARDCOPYBATCHFLG [Variable]

allows you to postpone printing messages until several can be done at once. Printing several messages at once helps to keep your short messages from being lost amongst other users’ long output, saves paper, and saves time because you don’t have to wait for individual messages to be formatted for printing.

When LAFITEHARDCOPYBATCHFLG is T, Lafite batches your hard-copy requests. It doesn’t actually print the messages until you do an Update, at which point Lafite sends them all to the printer in one batch. When you have hard copy pending, the Hardcopy command is speckled to remind you of this fact. The Update command has an additional choice, Do Hardcopy Only, in case you want to get your batched hard copy printed without doing an actual Update.

When batching hard copy, LAFITENEWPAGEFLG applies only to the messages selected at each Hardcopy invocation; each new set of messages starts on a new page, independent of the setting of LAFITENEWPAGEFLG. LAFITEHARDCOPYBATCHFLG is initially NIL.

LAFITEHARDCOPY.MIN.TOC [Variable]

is a positive number if non-NIL. Whenever Lafite is instructed to produce hard copy for more than LAFITEHARDCOPY.MIN.TOC messages, it also produces a table of contents as a cover page for the hard copy. Currently, this flag is noticed only if LAFITEHARDCOPYBATCHFLG is NIL. It is initially NIL.

MAILWATCHWAITTIME [Variable]

is the number of minutes between polling for new mail from your mail servers. It is initially set to five.

LAFITEFLUSHMAILFLG [Variable]

is initially T. If it is NIL, Lafite won't flush your in-box when retrieving new mail, so the mail will still be there when you invoke Get Mail again.

LAFITEIFFROMMETHENSEENFLG [Variable]

makes sure, if T, that messages sent from you are considered ‘‘seen’’ (and hence do not have the question mark), even though you have not yet displayed them. It is initially T.

LAFITEDISPLAYAFTERDELETEFLG [Variable]

can be set to T or ALWAYS. If T, Lafite displays the next message if you delete the message in the message display window and the next message is undeleted and unexamined (i.e., it is marked with a question mark). If ALWAYS, Lafite displays the next undeleted message even if it has already been seen. It is initially T.

LAFITEMOVETOCONFIRMFLG [Variable]

controls whether Lafite requires confirmation of the Move To command. If ALWAYS, all moves require confirmation; if LEFT, then only left-button moves (selecting the destination from a menu) require confirmation; if MIDDLE, then only middle-button moves (using the default Move To folder) require confirmation; if NIL, then no moves require confirmation. It is initially ALWAYS.

LAFITEBUFFERSIZE [Variable]

is the number of 512-character buffers used by the stream managing the file behind an open browser window. If you regularly receive very long messages, you might want to increase this to improve performance of Display followed by Hardcopy or Move To. Initially 20, which handles messages up to about 10,000 characters long.

LAFITEOUTBOXSIZE [Variable]

specifies the number of delivered messages to be retained in your out-box. As you send more messages, older ones fall off the end. Increasing this number gives you a longer ‘‘history’’ from which you can select and reedit old messages, but the desire for a longer history should be balanced with the knowledge that you are tying down the resources used by each of those messages. Setting LAFITEOUTBOXSIZE to zero or NIL disables the out-box feature: after delivery, messages completely vanish. LAFITEOUTBOXSIZE is initially two.

LAFITENEWMAILTUNE [Variable]

is a list of the form acceptable to the function PLAYTUNE or NIL, in which case it is ignored. It is played when Lafite discovers you have new mail waiting.

LAFITEGETMAILTUNE [Variable]

is a list of the form acceptable to the function PLAYTUNE or NIL, in which case it is ignored. It is played when a Get Mail command completes.

How to Add a New Message Form to Lafite

The normal way to add a new message form to Lafite is to edit an existing form (or build one from scratch) and save it using the Save Form menu item (see chapter 9, ‘‘Writing Messages’’). However, you can also provide message forms that compute the text at the time you request the form, as, for example, the Lisp Report does. To add your own items to the Message Forms menu, add a standard three-element menu item to the variable LAFITESPECIALFORMS and then set the variable LAFITEFORMSMENU to NIL (this is where the menu is cached). The three-element menu item should yield a LITATOM as its ‘‘value,’’ that atom being interpreted as follows:

1. If the atom has a function definition, the function is called (with no arguments) and the returned value (a string or a TEdit text stream) is used.

2. If the atom has a value, its value (a string or a TEdit text stream) is used.

3. Otherwise, a copy of the file by that name is used.

For example, if you wanted to add a message form that contained the date the user’s version of TEdit was made (as in the TEdit Report form), you could add

```
(‘‘TEdit Support’’ (QUOTE MAKETEDITFORM)
        ‘‘Make a form to report a problem with TEdit’’)
```

to LAFITESPECIALFORMS; MAKETEDITFORM could be defined to be

```
 [LAMBDA NIL
  (PROG (OUTSTREAM)
    (SETQ OUTSTREAM (OPENTEXTSTREAM “ ”))
    (printout OUTSTREAM ‘‘Subject: TEdit: >>Subject<<’’ T)
    (printout OUTSTREAM “To: ” TEDITSUPPORT T)
    (printout OUTSTREAM “cc: ” (USERNAME T)
    (printout OUTSTREAM “TEdit-System-Date:"
                                    TEDITSYSTEMDATE T T)
     (printout OUTSTREAM ‘‘>>Message<<’’ T)
     (RETURN OUTSTREAM]
```

where TEDITSUPPORT and TEDITSYSTEMDATE are variables set by TEdit. Lafite supplies one function to make this kind of message form easier to construct.

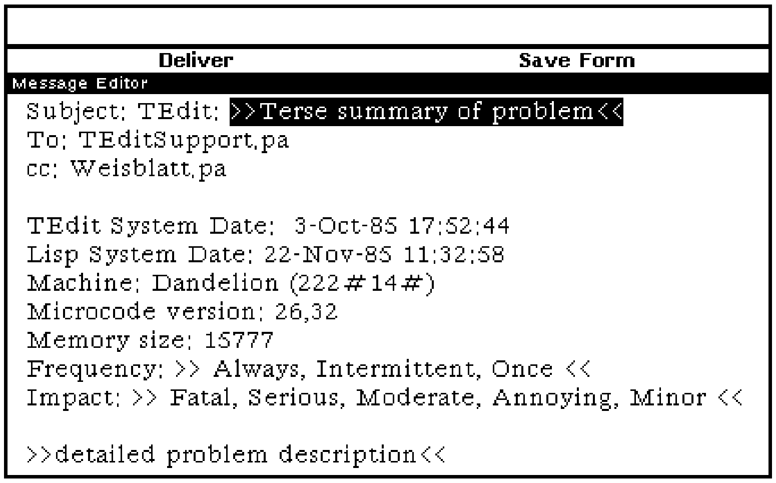
(MAKEXXXSUPPORTFORM *’SYSTEMNAME* *’ADDRESS* *’SYSTEMDATE*)[Function]

creates a message form (a TEdit stream) to be mailed to the maintainers of *SYSTEMNAME.* *SYSTEMNAME* is the name of the subsystem (a string); *SYSTEMDATE,* if non-NIL, is a date (string) of importance to include in the message; and *ADDRESS* is the mail system address of the intended recipient(s) or an association list in the same form as the variable LISPSUPPORT. If the *ADDRESS* is NIL, MAKEXXXSUPPORTFORM prints a message in the prompt window that the form is not supported, and returns NIL.

For example, MAKETEDITFORM is actually defined as

```
[LAMBDA NIL
   (MAKEXXXSUPPORTFORM
‘‘TEdit’’ TEDITSUPPORT TEDITSYSTEMDATE)]
```

and selecting TEdit Report in the Message Forms menu produces a form like the one in figure 29.



```
Subject: TEdit: >>Terse summary of problem<<
To: TEditSupport.pa
cc: Weisblatt.pa

TEdit System Date:  3-Oct-85 17:52:44
Lisp System Date: 22-Nov-85 11:32:58
Machine: Dandelion (222#14#)
Microcode version: 26,32
Memory size: 15777
Frequency: >> Always, Intermittent, Once <<
Impact: >> Fatal, Serious, Moderate, Annoying, Minor <<

>>detailed problem description<<
```

Figure 29. The TEdit report form that was created by following the instructions given above

14. USING LAFITE-RELATED LISP LIBRARY PACKAGES

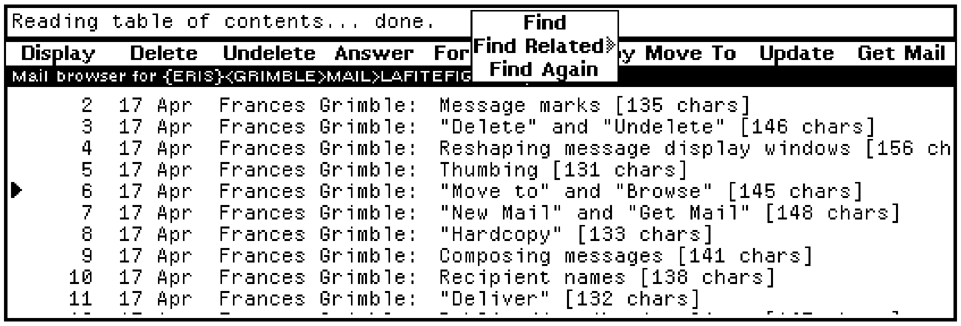
There are two separate packages that can be used with Lafite: Lafite Find and Mail Scavenger. Lafite Find helps you search for particular messages in a mail folder, and Mail Scavenger helps to restore mail files that have been damaged.

Lafite Find

Lafite Find is a package that helps you search for particular messages in a mail folder. The search is a simple string or keyword search that examines either specific header fields of a message (From, Subject) or the entire message. You can look for either one message or all messages matching a requested pattern.

How to Load Lafite Find

You load Lafite Find by typing (LOAD '*{FILESERVER}*<*DIRECTORY*>LAFITEFIND.DCOM). You can then bring up a menu of search commands by clicking on the black title bar of a Lafite browser window with the middle mouse button (see figure 30). Three commands are listed on this menu: Find, Find Related, and Find Again. The Find command is the general search command; the other two are special cases.



```
Reading table of contents... done.
Mail browser for {ERIS}<GRIMBLE>MAIL>LAFITEFIG
     2  17 Apr  Frances Grimble:  Message marks [135 chars]
     3  17 Apr  Frances Grimble:  "Delete" and "Undelete" [146 chars]
     4  17 Apr  Frances Grimble:  Reshaping message display windows [156 ch
     5  17 Apr  Frances Grimble:  Thumbing [131 chars]
▶    6  17 Apr  Frances Grimble:  "Move to" and "Browse" [145 chars]
     7  17 Apr  Frances Grimble:  "New Mail" and "Get Mail" [148 chars]
     8  17 Apr  Frances Grimble:  "Hardcopy" [133 chars]
     9  17 Apr  Frances Grimble:  Composing messages [141 chars]
    10  17 Apr  Frances Grimble:  Recipient names [138 chars]
    11  17 Apr  Frances Grimble:  "Deliver" [132 chars]
```

Figure 30. A browser window in which the user has brought up the top-level Lafite Find menu. You can bring up the menu anywhere on the title bar by clicking the middle mouse button

Search Direction Commands

If you select the Find command in this first menu, you are prompted with a menu to determine the direction of search. It lists four options: Find Next One, Find Next All, Find Previous One, and Find Previous All (see figure 31). To search forward in your mail folder one message at a time, use the Next One command. To search forward for all appropriate messages, use Next All. The Previous One and Previous All commands search backward. The Next searches all search forward from the last selected message in the browser; the Previous searches all search backward from the first selected message.

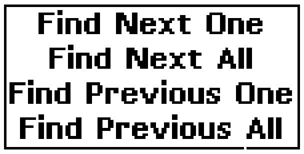


Figure 31. The search direction commands menu

The Find Next All and Find Previous All commands start the search with the currently selected message, unlike the Find Next One and Find Previous One commands, which omit the currently selected message. This is mainly so that the Find Related command includes the original message in the set of messages it selects.

Search Type Commands

Selecting any of the above commands brings up another menu, prompting for the type of search. This menu lists four options: From, Subject, Body, and Related (see figure 32). The From and Subject options enable you to search for messages by the content of their From or Subject fields, the Body search examines the entire message, and the Related search is a special kind of Subject search.

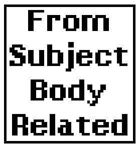


Figure 32. The search type commands menu

The From and Subject Searches

The From search looks at the From fields of messages sent by others and the To fields of messages sent by you. Thus, roughly speaking, From searches look at the same names you see in the browser window between the date and subject of each message. The Subject search examines the Subject fields of the messages in your mail folder.

The Body Search

The Body search can be used to locate a message by arbitrary strings in the message itself, rather than just in its From or Subject fields. For example, a Body search can locate messages that are addressed to (or cc'ed to) ‘‘Jones,’’ or that were sent in the month of ‘‘Feb,’’ or that mention ‘‘sugar’’ in their content. However, such searches are potentially much slower, since they have to search much more territory.

The Related Search

The Related search is a special kind of Subject search. It looks for messages whose subject is the same as the currently selected message, plus or minus the ‘‘Re: ’’ that the Answer command adds to a subject (see chapter 9, ‘‘Writing Messages’’). The easiest way to perform a Related search is to use the Find Related command in the initial menu, which is equivalent to the sequence Find, Find Next All, Related. If you want to do a Related search backward, the Find Related command has a subcommand Find Related Backward, which you can access by rolling the mouse cursor out of the menu to the right.

Find Related does not check that the messages it finds are really related to the selected message, merely that the subject matches the search string formed by taking the subject of the selected message and, if necessary, removing the initial ‘‘Re:.’’ Thus, this command may find too many messages.

Find Related is very useful for locating all the messages in an interchange where everyone used the Answer command to reply to the first (or subsequent) message.

How to Use Lafite Find

After you select the type of search (except for Related), you are prompted to type a string to search for. The search is case-insensitive; e.g., ‘‘Jones’’ matches both ‘‘JONES’’ and ‘‘jones.’’ The string searches are also substring searches, so you need not type the entire user name in a From search, just enough to uniquely identify the sender.

For example, to search for the next message sent by user Jones, click Find, then Find Next One, then From. Lafite then prompts you in the prompt region with ‘‘Find From string:.’’ Type the name of the sender (Jones) and then press the carriage return. If Lafite finds a message matching the string you typed, it selects the message and scrolls the browser, if necessary, to expose that message. If it finds no such message, it tells you so in the prompt region.

If you want to find all the messages from Jones, then follow the same procedure, but choose Find Next All instead of Find Next One. Lafite Find selects all the messages from Jones, and you can then cycle through them with the Display command, delete them all, move them all to a different folder, etc.

If the message found by a Find Next One or Find Previous One command is not the message you were looking for, or you want to locate the next such message, you can repeat the search by selecting the Find Again command in the initial (middle-button title bar) menu.

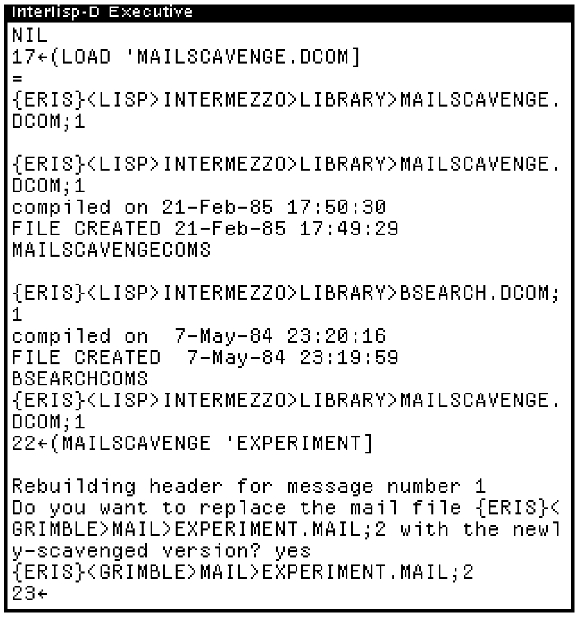
Mail Scavenger

The Lisp Library package Mail Scavenger is used to rebuild the internal pointers in a mail file that has been damaged (see chapter 12, ‘‘Troubleshooting Lafite Problems’’). When Lafite detects damage in a file, it usually reports ‘‘Can’t parse file’’ and terminates its Browse command. The simplest remedy is to scavenge the file with the Mail Scavenger, then browse the file again.

How to Use Mail Scavenger

To load the Mail Scavenger, type (LOAD ’*{FILESERVER}<DIRECTORY>*MAILSCAVENGE.DCOM) (see figure 33). To call the Mail Scavenger, type (MAILSCAVENGE ’*FILENAME*). This scavenges, or attempts to fix, the file named *FILENAME.* (*FILENAME* defaults to the extension MAIL and your Lafite directory, exactly as with Lafite’s Browse command.) Mail Scavenger will print some information in your executive window about what it is doing to correct the file, such as ‘‘Patching length field of header in message number 1.’’ It will then ask, ‘‘Do you want to replace *{FILESERVER}<DIRECTORY>MAILFILE* with the newly scavenged version?’’ If you answer Yes, the newly scavenged file will replace the damaged mail file. If you answer No, a new version of your mail file will be created. In this case, Mail Scavenger returns the name of the temporary file it wrote, ({DSK}<*DIRECTORY*>*FILENAME*.SCAVENGE), which you can then rename or delete as you wish. Ordinarily you should reply Yes, unless you are suspicious about Mail Scavenger's report on what it had to correct.

The finished mail file that Mail Scavenger produces should always be readable; i.e., Lafite will not complain about it. However, it is a good idea to browse the file and check any messages the Mail Scavenger mentioned correcting; these may be missing several characters or be malformed in other ways. You should also check neighboring messages—some of the characters in these messages might really be parts of other messages, and some messages may be duplicates. You may want to delete the damaged messages.



```
NIL
17←(LOAD 'MAILSCAVENGE.DCOM]
=
{ERIS}<LISP>INTERMEZZO>LIBRARY>MAILSCAVENGE.
DCOM;1

{ERIS}<LISP>INTERMEZZO>LIBRARY>MAILSCAVENGE.
DCOM;1
compiled on 21-Feb-85 17:50:30
FILE CREATED 21-Feb-85 17:49:29
MAILSCAVENGECOMS

{ERIS}<LISP>INTERMEZZO>LIBRARY>BSEARCH.DCOM;
1
compiled on  7-May-84 23:20:16
FILE CREATED  7-May-84 23:19:59
BSEARCHCOMS
{ERIS}<LISP>INTERMEZZO>LIBRARY>MAILSCAVENGE.
DCOM;1
22←(MAILSCAVENGE 'EXPERIMENT]

Rebuilding header for message number 1
Do you want to replace the mail file {ERIS}<
GRIMBLE>MAIL>EXPERIMENT.MAIL;2 with the newl
y-scavenged version? yes
{ERIS}<GRIMBLE>MAIL>EXPERIMENT.MAIL;2
23←
```

Figure 33. An executive window in which the user has loaded the Mail Scavenger and scavenged the folder Experiment.Mail

How to Scavenge Files in Place

Calling (MAILSCAVENGE.IN.PLACE *'FILENAME*) is similar to calling MAILSCAVENGE, except that it destructively scavenges the file in place. This is faster than MAILSCAVENGE, but you have to be brave and assume MAILSCAVENGE is not overwriting anything valuable as it scans the file.

Additional Arguments to MAILSCAVENGE

If you wish, you can call the MAILSCAVENGE function with the arguments *ERRORMSGSTREAM* and *TEMPDIR* as well as *FILENAME.* *ERRORMSGSTREAM* is the stream on which Mail Scavenger writes the information about what it is doing; the default is T. You can also specify the argument *TEMPDIR*, which is the host/directory on which Mail Scavenger writes its intermediate file. *TEMPDIR* defaults to {DSK} unless you are on a workstation without a local disk file system, in which case *TEMPDIR* defaults to the same directory as *FILENAME.*

CONTENTS

CONTENTS

A USER'S GUIDE TO LAFITE

XEROX

THE INTERLISP MAIL SYSTEM

3101489
January 1986

A USER’S GUIDE TO LAFITE

XEROX

THE INTERLISP MAIL SYSTEM

3101511
January 1986

LAFITEABBREV

By: Lennart Lövstrand
(Lovstrand.EuroPARC@Xerox.COM)

Requires: LAFITE, NSMAIL

INTERNAL

This document last edited on December 21, 1988.

INTRODUCTION

As the name suggests, LAFITEABBREV gives you the ability to refer to mail addresses using personal, easy to remember abbreviations. If you define (say) "Joe" to stand for "John Doe:Xyzzy North:ACME Corporation", then all you need to type to send a message to Joe is *Joe*. If you don't want to be bothered with his long address even when you read messages from him, you can optionally let the abbreviation apply both ways. When receiving messages from him (or where you both are recipients), his full address will be replaced by your abbreviation and thus only show as simple *Joe* in all header lines and the Lafite browsers.

In addition to this rather straight forward string-to-string text substitution, LAFITEABBREV will also do pattern matching using multiple wildcards in both the abbreviation and fully expanded strings. This is of course particularly useful for cases when many people share the same substring in their addresses, such as the NS domain. Say for example that Jane Doe worked at the same place as her brother. Then we could abbreviate "*:Xyzzy North:ACME Corporation" as "*:X:ACME" and refer to John as "John Doe:X:ACME" and Jane as "Jane Doe:X:ACME". In fact, we could even create the abbreviation "* Doe" ⇒ "* Doe:Xyzzy North:ACME Corporation" and refer to our friends as just "John Doe" and "Jane Doe" with the obvious expansions.

Abbreviations are purely personal; all other recipients of the message will see the real, expanded addresses. The abbreviations are on a strictly one-to-one mapping basis, ie. you can't implement private DLs by making an abbreviation expand into more than one address. As is the case with normal mail addresses, all abbreviations and expansions are case-insensitive.

EXAMPLES

An individual abbreviation:	Sir	⇒	Tom Moran:EuroPARC:RX
A group abbreviation:	* (APU)	⇒	*@MRC-APU.CAM.AC.UK:GV:Xerox
Hiding the local domain:	*	⇐	*:EuroPARC:RX
Hiding gateway syntax:	*@*.EDU	⇔	*%*:EDU:Xerox

MODULE EXPLANATION

LAFITEABBREV advises the Lafite NSMAIL routines that encode and decode string addresses to NSNAMEs. It operates transparently and automatically with respect to the rest of the mail system while being controlled by the variables mentioned below. The most central of these is the list of abbreviations itself, LAFITE.ABBREVS:

LAFITE.ABBREVS [Variable]

This variable contains all abbreviations and their corresponding real addresses. It is formed as a list of translations, where each translation is a list of (up to) four elements: the abbreviation, the full address, an optional direction, and an optional comment. Both the abbreviation and full address should be strings; the direction may be either of `:IN`, `:OUT`, `:BOTH`, `:NONE`, or left empty in which case it defaults to `:BOTH`. Finally, the comment, if present, should be in normal Interlisp format, ie. `(*` *text*`)`. It has no functional purpose and serves only as a memory aid. A full translation thus looks like:

(*abbrevation full-address direction comment*)

eg: `("Pete" "Piotr Kropotkin:RusskiPARK:ZeRoks" :OUT (* ; "Old chap"))`

which means that an address of *Pete* will be expanded to *Piotr Kropotkin:RusskiPARK:ZeRoks* whenever you send a message to him, but his full address will always be presented as such when receiving mail from him or whenever his address is mentioned in the header lines of a message you receive. For an explanation on translation directions, read more below.

The default list of translations is:

```
*@*     "*%*":GV:Xerox    :IN
*@*      *%*:GV:Xerox
*@*.*     *%*:*:Xerox     :IN
*.pa      *:PA:Xerox
```

This will present most ARPA Internet addresses in their proper form and also allow you to see Palo Alto Grapevine addresses as if you were on Grapevine. Note that the first rule above is necessary because some external GV addresses are being enclosed in double quotes by the GV/XNS Mail Gateway (for no appearant reason, one might add, since there exist no common conventions for parsing such constructs and it technically is malforming the address).

LAFITE.ABBREV.DIRECTIONS [Variable]

This variable controls the overall behaviour of LAFITEABBREV. It can take either of the following values:

`:IN`	Translate full addresses to abbreviations when receiving messages.
`:OUT`	Translate abbreviations to full addresses when sending messages.
`:BOTH`	Both of the above.
`:NONE`	Neither of the above, ie. don’t ever do any translations at all.

The setting of LAFITE.ABBREV.DIRECTIONS limits the scope of individual translation rules, so if you for example set LAFITE.ABBREV.DIRECTIONS to `:OUT`, no translations will ever be done on incoming messages, not even if a rule has an explict direction of `:IN` or `:BOTH`.

The default value of LAFITE.ABBREV.DIRECTIONS is `:BOTH`.

LAFITE.ABBREV.MOVE.GAZE.RIGHT [Variable]

A quite handy application of LAFITEABBREV is to present ARPA Internet (RFC-822) addresses in their proper form, ie. translating addresses like Jakobsson%Score.Stanford:EDU:Xerox to Jakobsson@Score.Stanford.EDU and vice versa. Unfortunately, since the pattern matching algorithm uses a strict left-to-right order, an address like user%host%foo.bar:EDU:Xerox would end up translated as user@host%foo.bar.EDU, which is not quite what was inteded. However, if LAFITE.ABBREV.MOVE.GAZE.RIGHT is set to a non-NIL value, translations resulting in a percent sign coming anywhere after an atsign will be automatically changed so that the rightmost percent sign is substituted for an atsign instead. Don't worry too much if you didn't understand any of the above; just think of it as applied magic (aka a *kludge*).

The default value of LAFITE.ABBREV.MOVE.GAZE.RIGHT is `T`.

LAFITE.ABBREV.TRACE [Variable]

Set this variable to T or a window of your choice (eg the PROMPTWINDOW) to trace all translations that are done for you. Reset to NIL to stop tracing.

The default value of LAFITE.ABBREV.TRACE is `NIL`.

(LAFITE.ABBREV *ADDRESS DIRECTION*) [Function]

This is the function that does the actual translations; it is used automatically by LAFITEABBREV's advices, but mentioned here if you want to test it manually or use it elsewhere. The address should be the string to expand/abbreviate and the direction either of `:IN` or `:OUT`, indicating whether this address is about to be sent or received. The result is the translated address.

(LAFITE.ABBREV.MATCH *PATTERN STRING TEMPLATE*) [Function]

This is a small, case insensitive pattern matching algorithm supporting multiple wildcards on both the pattern and the template side. Asterisks will match substrings of zero or more characters, anything else has to match literally for the comparison to succeed. The result is constructed using the template, if supplied. Some examples:

PATTERN	**STRING**	**TEMPLATE**	**RESULT**
"Foo"	"FOO"	NIL	"FOO"
"Foo"	"FOO"	"Bar"	"Bar"
"Foo*Bar"	"FOOBAZBAR"	NIL	"FOOBAZBAR"
"Foo*Bar"	"FOOBAZBAR"	"1*2"	"1BAZ2"
"Foo*B*ar"	"FOOBAZBAR"	"1*2*3"	"12AZB3"
"Foo*B*ar"	"FOOBAZBA"	"1*2*3"	NIL

BUGS

Only XNS is supported (ie. GV addresses aren't touched). Messages from you won't be presented as "To: <Recipients>" in the browser if you abbreviate your own name. No nice'n'cuddly user interface included.

LAFITE-INDENT

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

INTERNAL

This document last edited December 3, 1987.

INTRODUCTION

It is common practice in conversations conducted via electronic mail to include bits and pieces of previous messages in reply messages. The included text is indented to set it apart visually from the main body of the message. LAFITE-INDENT provides some support for this style of e–mail use.

USING LAFITE-INDENT

LAFITE-INDENT adds the item "Indent" to the default TEdit menu. Selecting the "Indent" item indents the current selection. In addition, it inserts carriage–returns in the text to ensure that no line in the selection is too long. The particular method of indentation is controlled by global variables described below. The "Indent" menu item also contains a number of subitems described below.

Indent [TEdit menu item]

This item will indent the current selection by prepending (the value of) *TEDIT-INDENT-STRING* to each line. To ensure that resulting text is properly formatted, lines will be broken to contain no more than *TEDIT-INDENT-LINE-LENGTH* characters. Existing carriage returns in the current selection may be replaced by spaces. For example the result of indenting the highlighted text below:

```
Date: 9 Sep 86 13:34:12
Subject: Eat?
To: JFrench
From: Wu

Hi Jane!
I was wondering if you might be able to make it over to our house tonight for dinner.
Carol's got a turkey that she wants to BBQ, Hank is coming over with a couple of bottle
of his home brew beer, and Diane's got a salad fresh from her garden.

Even if you can't make it for dinner, you might want to stop by to visit.  Remember
Crazy Tom from Berkeley?  He's in the area on a job interview and he'll be there.

Let me know soon...
```

```
Date: 9 Sep 86 13:34:12
Subject: Eat?
To: JFrench
From: Wu

    Hi Jane! I was wondering if you might be able to make it over
    to our house tonight for dinner.  Carol's got a turkey that she
    wants to BBQ, Hank is coming over with a couple of bottle of his
    home brew beer, and Diane's got a salad fresh from her garden.
```

```
        Even if you can’t make it for dinner, you might want to stop by
        to visit.  Remember Crazy Tom from Berkeley?  He’s in the area on a
        job interview and he’ll be there.

    Let me know soon...
```

Example 1: Before and after "Indent"

Indent & keep lines [TEdit menu item]

Like Indent, Indent & keep lines will indent the current selection. Unlike Indent, returns in the current selection are kept. This can be useful when indenting the headers of mail messages. For example the result of "Indent & keep lines" on the following text:

```
    Date: 9 Sep 86 13:34:12
    Subject: Eat?
    To: JFrench
    From: Wu

    Hi Jane!
```

```
        Date: 9 Sep 86 13:34:12
        Subject: Eat?
        To: JFrench
        From: Wu

    Hi Jane!
```

Example 2: Before and after "Indent & keep lines"

If Indent had been used instead of "Indent & keep lines", the lines would have been run together, resulting in:

```
        Date: 9 Sep 86 13:34:12 Subject: Eat? To: JFrench From: Wu
```

Set indent [TEdit menu item]

The "Set indent" message lets you change the current indentation string. It will prompt you for the new value of *TEDIT-INDENT-STRING*.

Unindent [TEdit menu item]

Unindent will remove all indentation from the current selection. In the process, it will replace all "extra" carriage returns by spaces. For example:

```
    Date: 9 Sep 86 13:34:12
    Subject: Eat?
    To: JFrench
    From: Wu

        Hi Jane! I was wondering if you might be able to make it over
        to our house tonight for dinner.  Carol’s got a turkey that she
        wants to BBQ, Hank is coming over with a couple of bottle of his
        home brew beer, and Diane’s got a salad fresh from her garden.

        Even if you can’t make it for dinner, you might want to stop by
        to visit.  Remember Crazy Tom from Berkeley?  He’s in the area on a
        job interview and he’ll be there.

    Let me know soon...
```

```
Date: 9 Sep 86 13:34:12
Subject: Eat?
To: JFrench
From: Wu

Hi Jane! I was wondering if you might be able to make it over to our house tonight for
dinner.  Carol's got a turkey that she wants to BBQ, Hank is coming over with a couple
of bottle of his home brew beer, and Diane's got a salad fresh from her garden.

Even if you can't make it for dinner, you might want to stop by to visit.  Remember
Crazy Tom from Berkeley?  He's in the area on a job interview and he'll be there.

Let me know soon...
```

Example 3: Before and after "Unindent"

Open line [TEdit menu item]

"Open line" will open a blank line at the current position in the textstream. Text following the current position maintain its distance from the left margin. For example:

```
Date: 9 Sep 86 13:51:20
Subject: Re: Eat?
To: Wu
From: JFrench

    Hi Jane! I was wondering if you might be able to make it over
    to our house tonight for dinner.  Carol's got a turkey that she
    wants to BBQ, Hank is coming over with a couple of bottle of his
    home brew beer, and Diane's got a salad fresh from her garden.
```

```
Date: 9 Sep 86 13:51:20
Subject: Re: Eat?
To: Wu
From: JFrench

    Hi Jane! I was wondering if you might be able to make it over
    to our house tonight for dinner.  Carol's got a turkey that she
    wants to BBQ, Hank is coming over with a couple of bottle
                                                          of his
    home brew beer, and Diane's got a salad fresh from her garden.
```

Example 4: Before and after "Open line"

Insert <RETURN>s [TEdit menu item]

This will replace TEdit's line breaks with real carriage returns (in the current selection). This way you know what line breaks recipients of your message will see.

Break long lines [TEdit menu item]

This item inserts <RETURN>s in the text, making sure that no line contains more than *TEDIT-INDENT-LINE-LENGTH* characters. Functionally, this is just like the "Indent" item, but without the indentation.

VARIABLES

The following variables can be changed to tailor how text is indented. They are defined as INITVARS in LAFITE-INDENT, so they can be given different values before or after loading the LAFITE-INDENT.

TEDIT-INDENT-STRING [GlobalVar]

This is the string that is used to indent each line of text. The default value is (the value of) `(ALLOCSTRING 4 " ");` a string consisting of four spaces.

TEDIT-INDENT-LINE-LENGTH [GlobalVar]

This is the maximum number of characters on a line that the indentation will allow. Extra line breaks will be added to ensure that no line has more than *TEDIT-INDENT-LINE-LENGTH* characters. The default value is `72`.

BUGS

Unfortunately, these operations are not (always) UNDOable.

LAFITETIMEDDELETE

By: John Maxwell (Maxwell.pa)

REQUIRES: LAFITE

INTERNAL

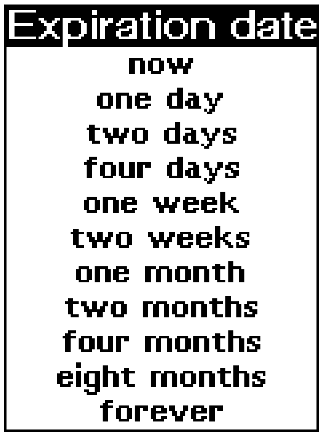
This document last edited on October 7, 1987.

INTRODUCTION

The package LafiteTimedDelete allows users to specify expiration dates on their mail messages as they read them. After a message has expired, it will be marked for deletion the next time the user invokes the command "Delete Expired Msgs". The package is useful for specifying deletion dates on dated information such as announcements of talks. It can also be used as a "sunset" clause on certain messages, saying in effect that if a message hasn’t been acted on by this date, mark it for deletion. Since the command for deleting expired messages only marks them for deletion and doesn’t actually delete them, the user can always intervene and save the message from deletion.

SETTING EXPIRATION DATES

To set an expiration date, simply select the message(s) that you wish to have deleted at some future time, and then click the "Delete" command on the Lafite browser with the middle button. This will produce a menu of durations like this:



Selecting a duration will determine how long until the message(s) expire, measured from the current day. If no selection is made, then the operation is aborted.

When a message is marked for future deletion, a number from 1 to 9 will appear in the mark field to the left of the message id. This number indicates how long until the message should expire, *measured from the date of the message*. The number is approximately the logarithm of the number of days until the message expires. Thus "1" means 1 day, "2" means 2 days, "3" means 4 days, "4" means 1 week, "5" means 2 weeks, "6" means 1 month, "7" means 2 months, "8" means 4 months, and "9" means 8 months.

Selecting the duration "now" is equivalent to normal deletion. Selecting the duration "forever" is equivalent to undeletion, with the side effect that all expiration marks are removed.

DELETING EXPIRED MESSAGES

To delete all of the messages that have expired, invoke the "Delete Expired Msgs" command in the browser’s middle button menu. (The middle button menu is obtained by holding down the middle button while over the black bar in the mail folder.) The program will then examine all of the messages in that folder, looking for messages that have expired. When it finds a message that has expired, it will mark it to be deleted. Finally, it will print in the browser’s prompt window the number of messages that it marked for deletion.

Messages expire at 12 noon on the Nth day from the date given in the date field of browser. Thus if on Wednesday, October 7th you mark a message sent that day to be deleted in two days, then the message would be deletable after 12 noon on Friday, October 9th. However, the messages don’t actually get marked for deletion until you invoke the command "Delete Expired Msgs". You usually only need to invoke this command once per day since no new messages will expire later on, so if you are unhappy with 12 noon as an expiration time, you can move it by only invoking the command after the day is over (or early in the morning the next day.)

CAVEATS

Since LAFITETIMEDDELETE uses the mark field of the message header to save the information about how long until the message expires, you may run into conflicts with other uses of the mark field. For instance, if you set the expiration date on a message and then forward it, you will lose the expiration information because the forwarder puts an "f" in the mark field. If you notice this, you can restore the mark manually by clicking the mark field and typing a new number.

Also, since LAFITETIMEDDELETE is limited in the number of distinctions that it can make in the time until expiration, it may set the expiration of a message to a time much later than you might want. For instance, suppose that you are reading a message that was mailed yesterday and you want to set its expiration date to one week from today. Since it can only record expirations from the date of the message and not from the current date, it must set the message to expire in eight days. If the program used the one week expiration mark, then the message would expire in seven days, or six days from today. Since that might expire the message before you intend, the program plays it safe and uses the two week expiration mark. A six day expiration might be OK with the user, but it would be awkward to ask him, especially if there were multiple messages selected. If the user is dissatisfied with the mark, he can always change it manually as described above.

Please send all comments, questions, and bug reports to Maxwell.pa.

MAILSCAVENGE

For Scavenging Mail Folders

The Lisp Library package `MAILSCAVENGE` is used to rebuild the internal pointers in a mail file that has been damaged. Lafite generally reports "Can't parse file" and terminates its Browse command when it detects damage in a file. The simplest remedy is to call `MAILSCAVENGE`, then browse the file again.

(`MAILSCAVENGE` *FILENAME ERRORMSGSTREAM TEMPDIR*) [Function]

> This function scavenges the file named *FILENAME*. *FILENAME* defaults to extension `MAIL` and your Lafite directory, exactly as with Lafite's Browse command. If *ERRORMSGSTREAM* is specified, it is a stream on which `MAILSCAVENGE` writes information about what it is doing to correct the file. *TEMPDIR* is a host/directory specification for where `MAILSCAVENGE` should write its intermediate file. *TEMPDIR* defaults to {`DSK`}, unless you are on an 1108 without a local disk file system, in which case it defaults to the same directory as *FILENAME*.
>
> When `MAILSCAVENGE` finishes, it asks if you want to replace the original file with the scavenged file. If you reply "No", `MAILSCAVENGE` returns the name of the temporary file it wrote, which you can then rename or delete as you wish. Ordinarily, you should reply "Yes", unless you find the report of what `MAILSCAVENGE` had to correct to be suspicious.

(`MAILSCAVENGE.IN.PLACE` *FILENAME ERRORMSGSTREAM*) [Function]

> This is similar to `MAILSCAVENGE`, but destructively scavenges the file in-place. This is faster than `MAILSCAVENGE`, but you have to be brave and assume `MAILSCAVENGE` is not overwriting anything valuable as it scans the file.

MIME decoding package, by Ron Kaplan, 2/2000.

This file recognizes and decodes email messages that conform to the MIME internet standard. It interfaces to Lafite so that it can parse the MIME headers and recognize various kinds of attachments. The attachments are represented in the message display window as an image-object box. Clicking on that box will let you extract the attachment and write it to a file on a directory which is obtained by evaluating the value of ATTACHMENTDIR. This is initialized to

```
        (CONCAT "{dsk}/tilde/"  (L-CASE (USERNAME)) "/attachments")
```

which, for example, would evaluate to "{dsk}/tilde/kaplan/attachments".

The MIME package tries to show text attachements in-line, and it offers a Print option for attachments (or even just included text) that it recognizes as Postscript.

The extracted files can be accessed by standard PC software packages. For certain Mac-derived application types, it tries to provide the appropriate PC extension so that double clicking will launch the appropriate application.

NSMAIL

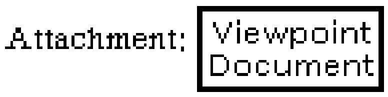
INTRODUCTION

The module NSMAIL implements the protocols to allow Lafite to be used to send and retrieve Xerox NS Mail. Load the file NSMAIL.LCOM. To run this in Lyric, you must have loaded the LispUsers module NSRANDOM as well (q.v. for important loading information). If you don’t have NSRANDOM loaded, you can’t use the "Put to file" command described below.

If you have both Grapevine and NSMAIL implementations loaded, you must set Lafite’s mode to NS. Use the "NS Mode" subitem underneath Lafite’s Quit command, or call (LAFITEMODE ’NS). You must also be a registered NS user, and have a mailbox.

ATTACHMENTS

The main difference between this and earlier versions of NSMAIL is that "attachments" are no longer left in your mailbox to be read later with, for example, Viewpoint. Instead, Lafite retrieves the entire attachment and encapsulates it into an image object that is enclosed as part of the text message, immediately following the header. A typical attachment appears in a mail message as:


Attachment: Viewpoint Document

If you click inside the object with any mouse button, you are offered a menu of things you can do with the attachment. The choices vary according to the type of attachment:

View as text	This brings up a window in which is displayed the raw content of the attachment as ascii bytes. Runs of non-ascii bytes are replaced by nulls to reduce the amount of garbage. Some attachments are utter gibberish, but some, such as Viewpoint documents and Interpress masters, contain sections that are plain text. With this command, you may be able to decide whether you care to do anything further with the attachment. (Sorry, there is no Viewpoint to TEdit converter, nor are there plans for one.)
Put to file	This prompts you for a file name, and creates a file to contain the attachment. The file must be on an NS file server for this command to be very useful; otherwise, information will be lost. Once the file is so stored, you can retrieve it from Viewpoint and manipulate it just as if you had originally retrieved it as mail in Viewpoint.

Send to Printer	This command is only available for attachments that are in the form of an Interpress master. The command prompts you for a printer (must be one that accepts Interpress, of course), and sends the attachment to it for printing.
Expand folder	This command is only available for attachments that are in the form of a "folder". A folder is a mechanism for collecting several objects into a single one. The Expand folder command splits the attachment up into its component objects, each of which can be manipulated in the same way as a top-level attachment. For example, if the folder contains an Interpress master, you can print it.

If you use the Put to file command on a folder, the name component of the file name you type will be treated as the name of a new subdirectory, and the components of the folder will appear as files in that subdirectory. For other types of attachments, Put to file (usually) produces an ordinary (non-directory) file.

Messages containing attachments are otherwise just like formatted messages—you can move them to other folders, and you can forward them (assuming the mail is received by another Lafite recipient and did not have to pass through a mail gateway).

There is currently no mechanism for creating your own attachments to end to other users.

MISCELLANY

If you prefer the old behavior of leaving the attachments behind in the mailbox, set the variable NSMAIL.LEAVE.ATTACHMENTS to T, but this use is discouraged. You must take care to regularly retrieve your mail from somewhere (such as Viewpoint) that will flush out all the mail; otherwise, the mail with attachments, whether you want them or not, accumulate on the server.

When in NS mode, Lafite will want your NS login identity. Normally, if your NS password differs from your default password, you will be prompted to login. You can also call (LOGIN '|NS::|) yourself to set your NS login.

You can freely intermix Grapevine and NS mail in the same mail folder if you like, but the Answer command always treats the selected message as if it were one in the current mode. So if you try to answer a Grapevine message while in NS mode, some confusion may result. Also, the status window always shows you mail status only of the current mode.

UNIXMAIL

By: Bob Bane (Bane.mv@envos.Xerox.com)

INTRODUCTION

UNIXMAIL is a new mail sending and receiving mode for Lafite. It sends mail via Unix hosts using the SMTP mail transfer protocol and can receive mail either by reading a Unix mail spool file or by calling the Berkeley mail program.

INSTALLATION

Turn Lafite off, load the file UNIXMAIL, make sure UNIXMAIL is configured appropriately (check the settings of the variables below, and make sure any other modules UNIXMAIL may need are loaded), then restart Lafite. If you are running Lafite on a machine that is isolated from the Xerox mail environment, you will probably want to set the variable LAFITE.USE.ALL.MODES to NIL and call (LAFITEMODE 'UNIX) before you turn Lafite back on.

CONFIGURING

See **SENDING MAIL** and **RECEIVING MAIL** below for the exact meanings of the variables you will be asked to set.

D-machines:

UNIXMAIL.SEND.MODE must be set to SOCKET and UNIXMAIL.SEND.HOST must be set to the name of a TCP host that will accept SMTP connections. UNIXMAIL.RECEIVE.MODE must be set to SPOOL and UNIXMAIL.SPOOL.FILE must be set to the pathname of your Unix mail spool file.

Unix-based emulators:

The default values of UNIXMAIL.SEND.MODE and UNIXMAIL.RECEIVE.MODE (PROCESS and SPOOL, respectively) will work if you normally send and receive mail from the machine where Medley is running.

OTHER MODULES YOU MAY NEED

UNIXMAIL may need other library modules to work. The modules needed vary depending on what hardware you are using:

D-machines:

TCP is mandatory for Unix sending and may be used for Unix receiving, NFS is optional for Unix receiving

Unix-based emulators:

one of TCPOPS or UNIXCOMM is mandatory for sending

SENDING MAIL

UNIXMAIL can send mail in one of two ways, depending on the setting of UNIXMAIL.SEND.MODE:

UNIXMAIL.SEND.MODE [Variable]

If its value is the atom PROCESS, UNIXMAIL will send mail by doing SMTP with a Unix process-stream, normally running `/usr/etc/mconnect`. This option only works on Medley running one of the Unix-based emulators.

If its value is the atom SOCKET, UNIXMAIL will send mail by doing SMTP with a TCP host. For this to work, an appropriate version of TCP must be loaded: either the TCP library module for D-machines or the TCPOPS library module for emulators that support it.

UNIXMAIL.SEND.MODE defaults to PROCESS.

Each of these send modes can be configured as well:

UNIXMAIL.SEND.PROCESS [Variable]

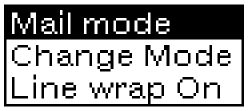
When UNIXMAIL.SEND.MODE is PROCESS, the value of this variable is the program run to create the SMTP process-stream. Initially the string `"/usr/etc/mconnect"`

UNIXMAIL.SEND.HOST [Variable]

When UNIXMAIL.SEND.MODE is SOCKET, the value of this variable is the name of the host UNIXMAIL will attempt to contact via TCP to open an SMTP stream over socket 25. Initially NIL; on a Unix-based emulator this means to try the machine Medley is running on. This variable must be set when running on a D-machine.

UNIXMAIL.WRAP.LINES [Variable]

This flag controls whether or not outgoing mail has its lines word-wrapped to a fixed length. It defaults to T, meaning word-wrapping is done. UNIXMAIL patches the Change Mode menu entry of the standard Lafite message form, adding an entry for toggling UNIXMAIL.WRAP.LINES:



UNIXMAIL.WRAP-LIMIT [Variable]

If UNIXMAIL.WRAP.LINES is true, this variable is the length in characters to which lines are wrapped. Default value is 72.

UNIXMAIL.TABWIDTH [Variable]

If UNIXMAIL.WRAP.LINES is true, this variable is the width tab characters are assumed to expand into for word-wrapping purposes. Default value is 8.

UNIXMAIL.RECIPIENT.PATTERNS [Variable]

This variable is a list of patterns that are applied to outgoing UNIXMAIL addresses; it can be used to catch bogus addresses and modify them before sending. List entries are of the form (*pattern* . *function*) where *pattern* is a list of arguments that will be passed along with the address to STRPOS; if STRPOS returns non-NIL, *function* is called with the address and the result replaces the address. For example, if mail from UUCP host `black-silicon` is arriving via path `mimsy!black-silicon`, but the address in its headers is missing the `mimsy`, this entry on UNIXMAIL.RECIPIENT.PATTERNS will add it back:

```
(("black-silicon!" NIL NIL T) LAMBDA (R) (CONCAT "mimsy!" R)
```

This means that whenever `(STRPOS "black-silicon!"` *`address`* `NIL NIL T)` returns non-NIL, *`address`* will have `"mimsy!"` prepended before the message is sent.

RECEIVING MAIL

UNIXMAIL can receive mail in one of two ways, depending on the setting of UNIXMAIL.RECEIVE.MODE:

UNIXMAIL.RECEIVE.MODE [Variable]

If its value is the atom SPOOL, UNIXMAIL will receive mail by reading a Unix mail spool file.

If its value is the atom MAILER, UNIXMAIL will receive mail by running a Berkeley mailer as a Unix process-stream, normally `/usr/ucb/mail`. This option only works on Medley running one of the

Unix-based emulators, and is a bit slower than SPOOL mode; it is primarily useful when you wish to occasionally switch between Lafite and the Berkeley mailer.

UNIXMAIL.RECEIVE.MODE defaults to SPOOL.

Each of these receive modes can be configured as well:

UNIXMAIL.RECEIVE.PROCESS [Variable]

When UNIXMAIL.RECEIVE.MODE is MAILER, the value of this variable is the program run to create the SMTP process-stream. Initially the string `"/usr/ucb/mail -N"`; the `-N` means to not print any banner or read any intialization file on starting the mailer.

UNIXMAIL.DONT.RECEIVE.STATUS [Variable]

When UNIXMAIL.RECEIVE.MODE is MAILER, the value of this variable is a set of message status letters; UNIXMAIL will leave behind any message whose status is included**.** Initially `""`, which means to read all messages regardless of status; another useful value would be `"O"` which means leave old messages behind.

UNIXMAIL.SPOOL.FILE [Variable]

When UNIXMAIL.RECEIVE.MODE is SPOOL, the value of this variable is the file UNIXMAIL will receive mail from. Any time this file has characters in it, Lafite will say you have new mail; when Lafite gets mail from this file, it will read all messages in the file and then set its size to zero. Initially NIL; on a Unix-based emulator this means to try the file `"{UNIX}/usr/spool/mail/`**`username`**`"`, where **`username`** is the value of (UNIX-USERNAME). To access a Unix mail spool file from a D-machine, it will probably be necessary to load and configure either the TCP or NFS modules and then set UNIXMAIL.SPOOL.FILE appropriately.

MAIKOCOLOR

Introduction

This module is the Envos Lisp software driver for running the COLOR module under Maiko (Maiko is the MACHINETYPE of Medley running on the Sun workstations). The machine independent functionality is provided by and documented with the COLOR module. There are no MAIKOCOLOR functions that the user needs to call directly. The user calls functions described in the COLOR documentation.

Requirements

In order to run COLOR, you need a Sun 3 or Sun 4 with a color display (CG 4), plus a color emulator (lde). COLOR will *not* run on a non-color emulator, attempting to use color capabilities will cause an error trap into URaid. You additionally need:

COLOR
LLCOLOR

Installation

MAIKOCOLOR may be loaded into a sysout running on any D-Machine (or non-color emulator), as long as color is not initialized. Thus an initial sysout can be made which runs on all systems, by loading COLOR, and then writing the sysout.

To install, simply (FILESLOAD ’MAIKOCOLOR), which will load all the additional files necessary for running color.

Initialization

To actually use color, the user must be running on a color capable emulator (lde).

To Initialize color running under Maiko, simply type

```
(COLORDISPLAY 'ON 'MAIKOCOLOR)
```

Lisp will permanently allocate a color "screen", and will attempt to map the color frame buffer to that screen.

At this point, the user should refer to the documentation for COLOR.

Known Bugs

NOTE: As this is currently unreleased software undergoing active development, this list of bugs should not be construed as being a limitation or defect of the final product. This list is only included to point out the current state of affairs of the software.

1. Color screen is never GC'ed

2. Color TEdit slower than B/W TEdit

3. Can't create big color windows

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 database of the results of the analyses it performs. Via a simple command language, you may interrogate the database, 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 database, 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 database 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` prints out the list of functions contained in the file `FOO` which use the variable `X` freely. The command `EDIT WHERE ANY CALLS 'ERROR` calls `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.

To use a keyword as a variable or function name, you must use a single quote in front of it, e.g., `.WHO SETS 'SETS`.

Note: Any MasterScope command may be followed by `OUTPUT` *FILENAME* to send output tothe 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 database. MasterScope does not reanalyze a function if it thinks it already has valid information about that function in its database. 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` automatically analyzes the functions in `MYFNS` if they have not already been analyzed.

Note also that only `EXPR` definitions are analyzed; that is, MasterScope does not analyze compiled code. If necessary, the definition is `DWIMIFY`ed before analysis. If there is no in-core definition for a function (either in the function definition cell or an `EXPR` property), MasterScope attempts 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 prints a message "(reading from *FILENAME*)". If no definition can be found at all, MasterScope prints 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 database. 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 database. `ERASE` by itself clears the entire database.

`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` prints `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 prints 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 calls 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
```

switches the first and second arguments to MYFN in every call to MYFN and prints the result. EDIT WHERE ANY ON MYFILE CALL ANY NOT @ GETD calls the editor on any expression involving a call to an undefined function.

Note that EDIT WHERE X SETS Y points only at those expressions where Y is actually set, and skips 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, is passed as a list of editor commands to be Executed.

For example,

```
EDIT ANY CALLING FN1 - (R FN1 FN2)
```

replaces 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 warns 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

`FOR` *VARIABLE SET I.S.TAIL* [MasterScope command]

This command provides a way of combining `CLISP` iterative statements with MasterScope. An iterative statement is 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)
```

prints 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 database; 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 `SMASH`es a variable if the function calls a destructive list operation (`RPLACA`, `RPLACD`, `DREMOVE`, `SORT`, etc.) on the value of that variable. MasterScope also finds 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 is noted as `SMASH`ing `X`.

If the function contains a sequence `(SETQ Y X)`, `(RPLACA Y T)`, then `Y` is noted as being `SMASH`ed, but not `X`.

`TEST` [MasterScope relation]

A variable is `TEST`ed 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 `USE`s `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 database 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 SMASH`es 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` returns 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 `ESC`apes; 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` prints 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 FOO`VARS`.

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, prints 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) prints 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 to be displayed, while the second restricts the paths to be traced; i.e., the command `SHOW PATHS FROM X TO Y` traces 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 analyzes 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` does 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` is printed, but the tree beyond it is not expanded. The functions in an `AVOIDING` set are not printed at all.

For example,

```
SHOW PATHS FROM ANY ON FILE1 NOTRACE ON FILE2
```

displays the tree of calls eminating from `FILE1`, but does 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 are displayed. `SEPARATE` can be used to guarantee that certain functions 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 lowercase 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 prints the message

```
Sorry, that isn’t implemented!
```

and generates an error.

If the command requires some functions having been analyzed (e.g., the command `WHO CALLS X`) and the database is empty, MasterScope prints the message

```
Sorry, no functions have been analyzed!
```

and generates an error.

Macro Expansion

As part of analysis, MasterScope expands 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 searches to find a macro definition. Only the kinds of macros that appear on `MSMACROPROPS` are expanded. All others are 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 knows 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 does 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 database 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 prints 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 notes 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 database 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 TEMPLATE* `)` [in MasterScope template]

Evaluate *EXPRESSION* at analysis time (the variable `EXPR` is 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 `APPLY`d 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 uses 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 marks 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 database are marked as calling *FN*, they are marked as needing reanalysis.

Updating the MasterScope Database

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 database 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`, enters into an Executive in which you may enter commands. If *COMMAND* is not `NIL`, the command is interpreted and MASTERSCOPE returns 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` also works 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` performs a one-time analysis of *FN*. Otherwise (i.e., if *USEDATABASE* is non-`NIL` and *FN* a function name), `CALLS` uses the information in MasterScope's database (*FN* is 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* is 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 database:

`(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` first performs `(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` returns the list of atoms which have *RELATION* with *any* other item; i.e., it answers the question `WHO` *RELATION*`S 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 `LOAD`ed or `UNSAVE`d (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 analyzes *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 reanalyzes *FN* even if MasterScope thinks it has a valid analysis in the database.

(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 database on the current output file in a LOADable form. If *FNLST* is not NIL, DUMPDATABASE only dumps 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 database. If a MasterScope database already exists when a DATABASE file is loaded, the database on the file is merged with the one in memory.

Note: Functions whose definitions are different from their definition when the database 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 databases 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 database), MasterScope alerts 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 still thinks 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 database 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 database considerably, to the point where it is reasonable to maintain a MasterScope database 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 database, MasterScope performs unions of the hash values.

For example, to answer `FOO BINDS WHO`, MasterScope looks 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 `GETHASH`s on the appropriate hash tables and `UNION`ing 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 database 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 prints out a period. (If an error in the function is detected, "?" is printed instead.) If MSPRINTFLG is a number *N*, MasterScope prints 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 database 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 prints 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.

[This page intentionally left blank]

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 whichperforms 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 is 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 matches 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 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). 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`.

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`.

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 * is 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 using `$` results in a more efficient search than that using `--`. However, `($ 'A 'B $)` does not match with `(X Y Z A M O A B C)`, but `(-- 'A 'B $)` does.

Essentially, once a pattern following a `$` matches, the `$` never resumes searching, whereas `--` produces a translation that always continues 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 `$` is 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` matches 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 requires 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 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))` sets `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. For example, `(MATCH X WITH (Y_$1 =Y --))` does 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` is 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)))
```

matches 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)) replaces 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 is 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 is (A 1 2 3 D E F), and FOO is 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)))
```

Limitation

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 results 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]
```

[This page intentionally left blank]

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 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 database.

Most of the List commands enumerate items in the database 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) match your pattern against an alias, so you may want to use the List Aliases command if you do not 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 database. 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 database.
`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 database. 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 database. 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 database.
`Remove Registered Object`	Removes a specified object and all its properties from the database. 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`
`Add Friend`
`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`	

`Remove Friend`	
`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 database, 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 database. Check that the spelling and the domain are correct.
`IllegalOrganization`	
`IllegalDomain`	
`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]

POSTSCRIPTSTREAM

By: Matt Heffron (mheffron@orion.cf.uci.edu)

INTRODUCTION

The PostScript package defines a set of imageops for printers which understand the PostScript page description language by Adobe. At Beckman we have successfully used TEdit, Sketch, LISTFILES, and HARDCOPYW to an Apple LaserWriter and an AST TurboLaser PS. The PostScript imagestream driver installs itself when it is loaded. All symbols in the PostScript driver are located in the INTERLISP: package.

VARIABLES

POSTSCRIPT.FONT.ALIST [InitVariable]

POSTSCRIPT.FONT.ALIST is an ALIST mapping Xerox Lisp font names into the root names of PostScript font files. It is also used for font family coercions. The default value should be acceptable for any of the fonts which are built into the Apple Laserwriter.

POSTSCRIPTFONTDIRECTORIES [InitVariable]

POSTSCRIPTFONTDIRECTORIES is the list of directories where the PostScript .PSCFONT font files can be found. The default value is: ("{DSK}/usr/local/lde/fonts/postscript/") on a Sun or IBM workstation and ("{DSK}<LISPFILES>FONTS>PSC>") for other cases .

POSTSCRIPT.DEFAULT.PAGEREGION [InitVariable]

POSTSCRIPT.DEFAULT.PAGEREGION indicates the area of the page to use for text file listings (i.e. LISTFILES). It is in units of 100'ths of points. The default value is: (4800 4800 52800 70800), which gives left and bottom margins of 0.75 inch and top and right margins of 0.5 inch on 8.5 x 11 paper.

POSTSCRIPT.PAGEREGIONS [InitVariable]

POSTSCRIPT.PAGEREGIONS is an ALIST mapping pagetypes into paper size and actual imageable area on the page. By default, it knows about LETTER, LEGAL, and NOTE pagetypes, and the corresponding sizes and imageable areas for the Apple Laserwriter. Others can be defined by the user by adding the appropriate entries onto this ALIST.

POSTSCRIPT.PAGETYPE [InitVariable]

POSTSCRIPT.PAGETYPE is used by OPENIMAGESTREAM to lookup the paper size and actual imageable area of the page in POSTSCRIPT.PAGEREGIONS to determine the initial margins. This value can be overridden with the PAGETYPE or PAPERTYPE options in the OPENIMAGESTREAM call. The name of the type of page selected is NOT passed through to the PostScript output.

\POSTSCRIPT.MAX.WILD.FONTSIZE [InitVariable]

\POSTSCRIPT.MAX.WILD.FONTSIZE indicates the maximum point size that should be returned from FONTSAVAILABLE when the SIZE argument is wild (i.e. *). All integer pointsizes from 1 to \POSTSCRIPT.MAX.WILD.FONTSIZE will be indicated as available. The default value is: 72.

POSTSCRIPT.PREFER.LANDSCAPE [InitVariable]

POSTSCRIPT.PREFER.LANDSCAPE indicates if the OPENIMAGESTREAM method should default the orientation of output files to LANDSCAPE. It can have one of three values: NIL, T, or ASK. NIL means prefer portrait orientation output, T means prefer landscape, and ASK says to bring up a menu to ask the preferred orientation if it wasn't explicitly indicated in the OPENIMAGESTREAM call (with the ROTATION option). The default value is: NIL. An item (PS Orientation) is added to the Background Menu to let you change the value of this variable.

POSTSCRIPT.TEXTFILE.LANDSCAPE [InitVariable]

POSTSCRIPT.TEXTFILE.LANDSCAPE indicates if the printing of TEXT files (e.g. LISTFILES, ...) should force the orientation of output files to LANDSCAPE. When it is non-NIL the orientation of output files is forced to LANDSCAPE. (There is no ASK option here.) The default value is: NIL.

POSTSCRIPT.BITMAP.SCALE [InitVariable]

POSTSCRIPT.BITMAP.SCALE specifies an independent scale factor for display of bitmap images (e.g. window hardcopies). Values less than 1 will reduce the image size. (I.e. a value of 0.5 will give a half size bitmap image.) The position of the scaled bitmap will still have the SAME lower-left corner (i.e. the scaled bitmap is not centered in the region of the full size bitmap image). The default value is: 1.

HINT

> Setting POSTSCRIPT.BITMAP.SCALE to 0.96, instead of 1, will give cleaner BITMAP images on a 300 dpi printer. (This corrects for the 72 ppi imagestream *vs*. the 75 dpi printer, using 4x4 device dots per bitmap pixel.) Also, values of 0.24, 0.48 and 0.72, instead of 0.25, 0.5 and 0.75, will also give cleaner images for reduced size output. In general, use integer multiples of 0.24 for a 300 dpi printer.

POSTSCRIPT.TEXTURE.SCALE [InitVariable]

POSTSCRIPT.TEXTURE.SCALE specifies an independent scale for the display of bitmap textures. The value represents the number of device space units per texture unit (bitmap bit). The default value is 4, which represents each bit of the texture as a 4x4 block, so that textures are approximately the same resolution as on the screen (for 300 dpi output devices, such as the Apple Laserwriter).

The PostScript package extends the allowed representations of a texture, beyond 16-bit FIXP and 16x16 bitmap, to ANY square bitmap. (If the bitmap is not square, its longer edge is truncated from the top or right to make it square.) Use this feature with caution, as large bitmap textures, or sizes other than multiples of 16 bits square, require large amounts of storage in the PostScript interpreter (in the printer controller), and can cause limitcheck errors when actually printing.

Anywhere that a texture or color can be used on an imagestream or in the specification of a BRUSH, you can instead give a FLOATP between 0.0 and 1.0 (inclusive) to represent a PostScript halftone gray shade. (0.0 is black and 1.0 is white. Specifically, the value sets the brightness of the shade.) The value you specify will not be range checked, and will be passed directly through to the PostScript setgray operator. (E.g. you can pass 0.33 as the color to DRAWLINE to get a dark gray line with approximately 67% of the pixels in the line black.)

POSTSCRIPT.IMAGESIZEFACTOR [InitVariable]

POSTSCRIPT.IMAGESIZEFACTOR specifies an independent factor to change the overall size of the printed image. This re-sizing affects the entire printed output (specifically, it superimposes its effects upon those of POSTSCRIPT.BITMAP.SCALE and POSTSCRIPT.TEXTURE.SCALE). Values greater than 1 enlarge the printed image, and values less than 1 reduce it. An invalid POSTSCRIPT.IMAGESIZEFACTOR (i.e. not a positive, non-zero number) will use a value of 1. The BITMAPSCALE function for the POSTSCRIPT printer type does NOT consider the POSTSCRIPT.IMAGESIZEFACTOR when determining the scale factor for a bitmap.

MISCELLANEOUS

The SCALE of a PostScript imagestream is 100. This is to allow enough resolution in the width information for fonts to enable TEdit to correctly fill and justify text.

The first time any PostScript imagestream is created (even if only to hardcopy a bitmap or window) the DEFAULTFONT is instantiated (unless a FONTS option was given to the OPENIMAGESTREAM, in which case the initial font for the imagestream will be set to that font, or to the CAR if a list).

The PostScript imagestream method for FILLPOLYGON uses the global variable FILL.WRULE as the default value for the WINDINGNUMBER argument. (This is the same variable which is used by the DISPLAY imagestream method for FILLPOLYGON.)

The PostScript imagestream method for OPENIMAGESTREAM (and, therefore, SEND.FILE.TO.PRINTER), supports an IMAGESIZEFACTOR option to change the size of the printed image. The IMAGESIZEFACTOR re-sizing is combined with the POSTSCRIPT.IMAGESIZEFACTOR to produce an overall re-sizing of the printed image. A HEADING option is also supported to give a running header on each page of output. The value of the HEADING option is printed at the top left of the page, followed by "Page " and the appropriate page number. They are printed in the DEFAULTFONT (unless a FONTS option was given to the OPENIMAGESTREAM, in which case it will be that font, or to the CAR if a list).

The PostScript package is contained in the files: POSTSCRIPTSTREAM.LCOM & PS-SEND.LCOM, with the source in the files: POSTSCRIPTSTREAM & PS-SEND. The module PS-SEND.LCOM is required and will be loaded automatically when POSTSCRIPTSTREAM.LCOM is loaded. It contains the function which is called by SEND.FILE.TO.PRINTER to actually transmit the file to the printer. It is, by its nature, quite site specific, so it is in a separate file to make modifying it for any site relatively simple. System record declarations required to compile POSTSCRIPTSTREAM can be found in EXPORTS.ALL.

I'm pretty sure that the output generated by the PostScript imageops fully conforms to the Adobe Systems Document Structuring Conventions, Version 2.0, January 31, 1987.

Including Other PostScript Operations

If you wish to insert your own specific PostScript operations into a PostScript imagestream, you can do so with the following functions:

(POSTSCRIPT.OUTSTR *STREAM STRING*) [Function]

POSTSCRIPT.OUTSTR outputs a string or value to the imagestream. *STREAM* must be an open PostScript imagestream. *STRING* is the value to output (STRINGP and LITATOM are most efficient, but any value can be output (its PRIN1 pname is used)).

(POSTSCRIPT.PUTCOMMAND *STREAM STRING ... STRING*) [NoSpread Function]

POSTSCRIPT.PUTCOMMAND is more general for sequences of commands and values. It calls POSTSCRIPT.OUTSTR repeatedly to output each of the *STRING* arguments to *STREAM*.

(\POSTSCRIPT.OUTCHARFN *STREAM CHAR*) [Function]

\POSTSCRIPT.OUTCHARFN is used to output the characters forming the text of a PostScript string (e.g. the argument to a show or charpath operator). *STREAM* is the open PostScript imagestream to output to, and *CHAR* is the CHARCODE of the character to output. The ***/*** (slash), **(** and **)** (parenthesis) characters will be quoted with ***/***, and characters with ASCII values less than 32 (space) or greater than 126 (tilde) will be output as ***/*nnn** (in octal). \POSTSCRIPT.OUTCHARFN will output the **(** character to open the string, if necessary. Use POSTSCRIPT.CLOSESTRING (below) to close the string.

(POSTSCRIPT.CLOSESTRING *STREAM*) [Function]

POSTSCRIPT.CLOSESTRING closes a PostScript string (e.g. the argument to a show or charpath operator). *STREAM* is the open PostScript imagestream. It is important to use POSTSCRIPT.CLOSESTRING to output the **)** character to close the string, because it also clears the stream state flag that indicates that a string is in progress (otherwise, the next POSTSCRIPT.PUTCOMMAND would output the commands to close the string and show it).

Warning

Do not attempt to create a PostScript font larger than about 600 points, as much of Interlisp's font information is stored in SMALLP integers, and too large a font would overflow the font's height, or the width for any of the wider characters. (I know that 600 points is a ridiculously large limit (about 8.3 inches), but I thought I'd better mention it, or someone might try it!)

Changes from the Initial Medley Release

This second Medley release of the PostScript imagestream driver includes some performance enhancements when writing bitmaps to the output, some SUN-specific code (from Will Snow of envos), implementation of the SCALEDBITBLT, DSPROTATE, and DSPTRANSLATE operations, and a lot of performance enhancements (many thanks to Tom Lipkis of Savoir).

Changes from the Lyric Release

The Medley release of this PostScript imagestream driver changed the default value of POSTSCRIPT.TEXTFILE.LANDSCAPE from T to NIL. It also added the support for the HEADING option.

Known Problems/Limitations

The output generated for a PostScript imagestream is rather brute force. It isn't particularly careful to generate the smallest output file for a given sequence of operations. Specifically, it often generates extra end-of-lines between PostScript operator sequences (this has no effect on the printed output, only on the file size).

Using BITMAPs or Functions as BRUSH arguments to the curve drawing functions is not supported, nor is using a non-ROUND BRUSH with DRAWCIRCLE or DRAWELLIPSE.

The implementation of DSPROTATE accepts ROTATION argument values of 0 and 90 (any non-NIL, non-zero value is converted to 90). A value of 0 converts the page orientation to Portrait, and 90 converts the page orientation to Landscape. These conversions perform the translations necessary to keep the clipping region on the page. (This may or may not be the right thing to do, but since DSPROTATE is undocumented in what it should do, this is what the PostScript driver does).

There is no support for NS character sets other than 0, and there is no translation of the character code values from NS encoding to PostScript encoding.

There is no support for color.

\POSTSCRIPT.OUTCHARFN is pretty wimpy in its handling of TAB characters. It just moves to the next multiple of (eight times the average character width of the current font) from the current left margin.

I haven't yet documented how to build the .PSCFONT files from .AFM files for new fonts that become available.

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
- Functions typed into the exec window (see below)

Functions

DEFAULTPRINTINGHOST	[Variable]
PRESSFONTWIDTHFILES	[Variable]
SEND.FILE.TO.PRINTER	[Function]
LISTFILES	[Function]

(For details about all the above, see the *IRM*.)

Limitations

PRESS does not support NS characters or NS fonts.

[This page intentionally left blank]

READNUMBER

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 is put there; otherwise it is put at the cursor.

If *MSGFONT* is given, `MSG` is printed in it. If *MSGFONT* is `NIL`, `DEFAULTFONT` is used.

If *DIGITFONT* is given, the labels on the keys is printed in that font. If *DIGITFONT* is `NIL`, `BOLDFONT` is used.

If *INCLUDEABORTFLG* is non-`NIL`, the menu also includes an abort key (abt). If the abort key is pressed, `RNUMBER` returns `NIL`.

Note: If this option is set, you will not be able to use the backspace to correct mistakes. You will have to use CLEAR and begin the number again.

If *FLOATINGPTFLG* is non-`NIL`, the menu includes a decimal point, and the value returned may be a floating point number.

If *POSITIVEONLYFLG* is non-`NIL`, the menu does not include a +/- key (-) and you can only input positive numbers (but see `ACCEPTTYPEINFLG`).

If *ACCEPTTYPEINFLG* is non-`NIL`, the menu also responds 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.

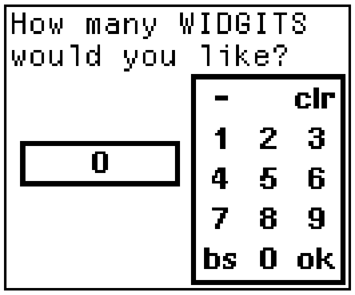
(NUMBERPAD.READ *NUMBERPAD/READER ACCEPTTYPEINFLG*) [Function]

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

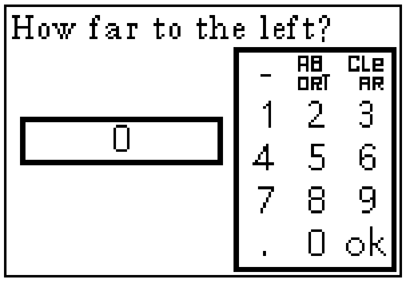
```
(RNUMBER "How many WIDGITS would you like?")
```

results in the following pop-up menu:



```
(RNUMBER "How far to the left?") NIL '(CLASSIC 12) '(MODERN 14) T T)
```

results in the following pop-up menu:



Limitations

If you choose both FLOATNGPOINTFLG and INCLUDEABORTFLG, there is no room for the backspace key (see the illustration above). Correct your input by selecting CLEAR and starting over. However, if ACCEPTTYPEINFLG is T, you can use the keyboard's backspace key.

[This page intentionally left blank]

RS232

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.

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).
`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 maximum 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.

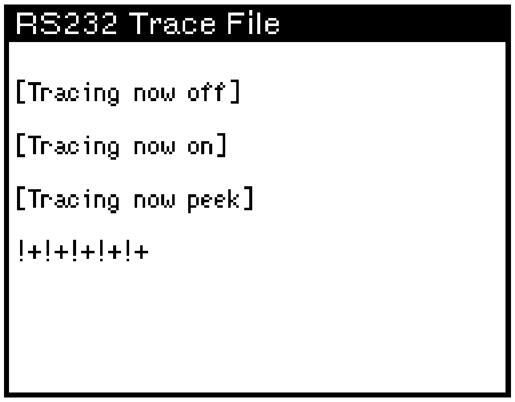
In addition, if *NEWVALUE* is supplied, the current setting of status reporting is changed to *NEWVALUE*.

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` opens 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` turns off the tracing. Clicking the left mouse button in the trace window will cycle between the modes.



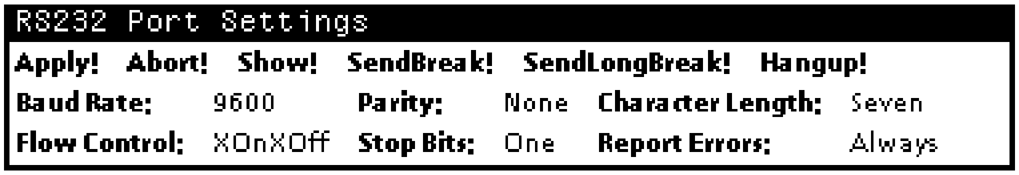
```
[Tracing now off]

[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.



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:

1. Load RS232CHAT.LCOM on both machines.
2. Call RS232C.INIT to set up parameters.
3. 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.

[This page intentionally left blank]

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.

[This page intentionally left blank]

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 oly 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

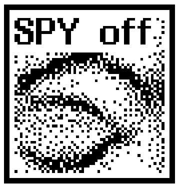
Load `SPY.LCOM` and the required `.LCOM` modules from the library.

User Interface

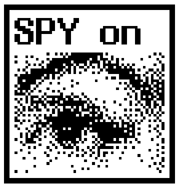
`(SPY.BUTTON` *`POS`*`)` [Function]

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 turns on sampling, and the window looks like this:



Clicking on it again turns off sampling, and displays the results `(SPY.TREE 10)`. This is the simplest way of spying on operations. (See `SPY.TREE` below.)

Note: You cannot 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)
```

enables Control-C (or any other character you specify) as an interrupt which turns spying on and off, the same as clicking on the Spy button. (If the Spy button is visible, it responds to the interrupt.) Then, a Control-C turns on spying, and another one turns 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.

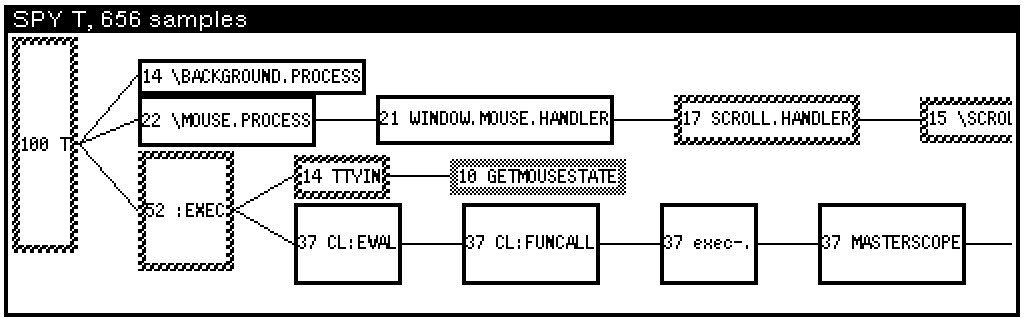
THRESHOLD is a percentage (defaults to 0). If a function's contribution to the total elapsed time is lower than the threshold percentage, it is not displayed.

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

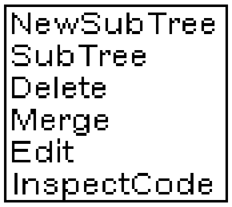
You can point at any of the nodes.

Right Button Operation

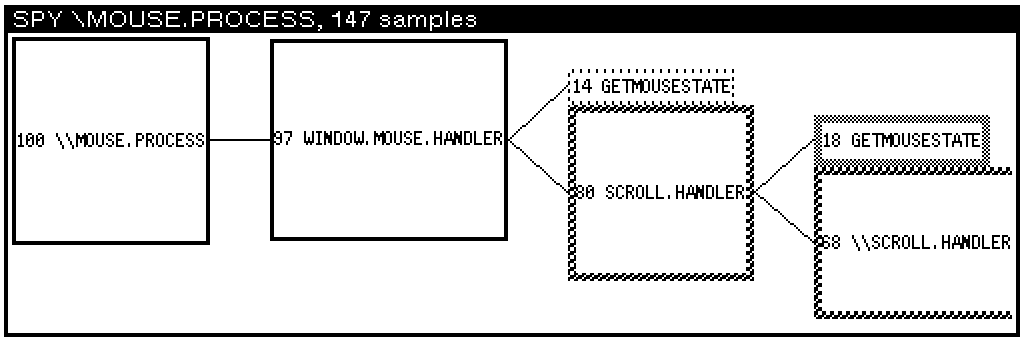
If you right-click on a node, the window title will change to show the function name, and the individual and cumulative percentages. The first number is the individual total and the second is the cumulative. If the right-click is not on a node, the title will change to show the name of the top frame 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:



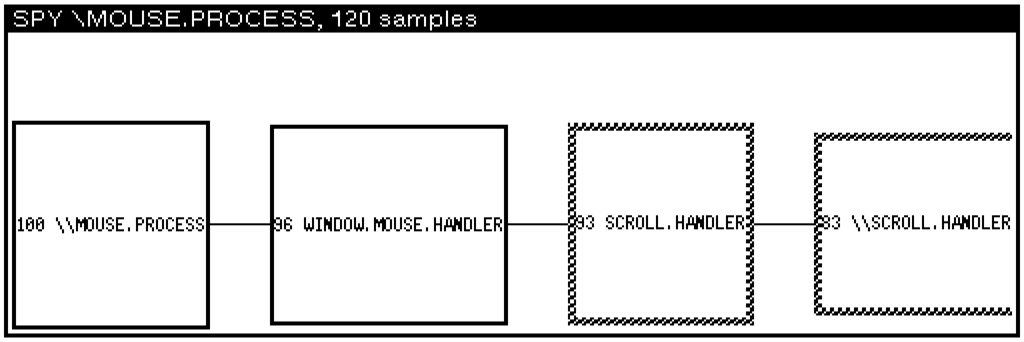
NewSubTree Creates another Spy window that includes data only from this node and its descendents (with the tree rooted at the selected node). Suppose that you were only interested in the actions that you had invoked with the mouse. You can left-click `\MOUSE.PROCESS` and select the NewSubTree option. 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:

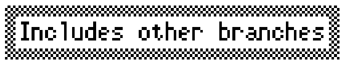
Legend	Displays SPY border interpretation.
Inspect	Allows for the inspection of the current parameter settings for the Spy window.
SetThreshold	Sets the threshold for displaying a node. Any node whose percentage is below the threshold won't show up (unless it is needed to connect the graph together). 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

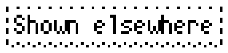
If you press the left, or middle button while the control key is down, you will get the same menu, but the action will be deferred until you next use the left/middle button. For example, you can delete several nodes and then do one update.

Merged Nodes

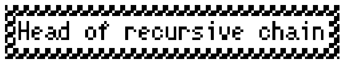
If two nodes are merged, then the merged node will include the time (and descendents) of the other. The display of a merged node is different; it is shown with a thick gray border; e.g. ,



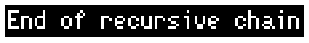
The time in a merged node is the sum of the times for all occurrences. Other calls to the same function may show up as ghost boxes; e.g.



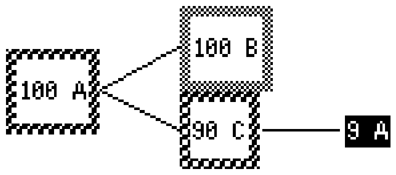
The case where a function is merged with a recursive call to itself is handled specially: the head of the recursion is marked with a wider checkered border:



and the tail of the recursion is shown in reverse video:



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.

[This page intentionally left blank]

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.

[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.

Known Problems

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

[This page intentionally left blank]

UNICODE

By Ron Kaplan

This document was last edited in July 2023.

The `UNICODE` library package defines external file formats that enable Medley to read and write files where 16 bit character codes are represented as UTF-8 byte sequences or big-endian UTF16 byte-pairs. It also provides for character codes to be converted (on reading) from Unicode codes to equivalent codes in the Medley-internal Xerox Character Code Standard (`XCCS`) and (on writing) from XCCS codes to equivalent Unicode codes.

UTF-8 External formats

Four external formats are defined when the package is loaded:

`:UTF-8`	codes are represented as UTF-8 byte sequences and XCCS/Unicode character conversion takes place.
`:UTF-16BE`	codes are represented as 2-byte pairs, with the high order byte appearing first in the file, and characters are converted.

The two other external formats translate byte sequences into codes, but do not translate the codes. These allow Medley to see and process characters in their native encoding.

`:UTF-8-RAW`	codes are represented as UTF-8 byte sequences, but character conversion does not take place.
`:UTF-16BE-RAW`	codes are represented as big-ending 2-byte pairs but there is no conversion.

These formats all define the end-of-line convention (mostly for writing) for the external files according to the variable `EXTERNALEOL` (`LF`, `CR`, `CRLF`), with `LF` the default.

The external format can be specified as a parameter when a stream is opened:

```
(OPENSTREAM 'foo.txt 'INPUT 'OLD '((EXTERNALFORMAT :UTF-8)))
```

```
(CL:OPEN 'foo.txt :DIRECTION :INPUT :EXTERNAL-FORMAT :UTF-8)
```

The function `STREAMPROP` obtains or changes the external format of an open stream:

```
(STREAMPROP stream 'EXTERNALFORMAT) -> :XCCS
```

```
(STREAMPROP stream 'EXTERNALFORMAT :UTF-8) -> :XCCS
```

In the latter case, the stream's format is changed to `:UTF-8` and the previous value is returned, in this example it is Medley's historical default format `:XCCS`.

Entries can be placed on the variable `*DEFAULT-EXTERNALFORMATS*` to change the external format that is set by default when a file is opened on a particular device. Loading UNICODE executes

```
(PUSH *DEFAULT-EXTERNALFORMATS* '(UNIX :UTF-8))
```

so that all files opened (by OPENSTREAM, CL:OPEN, etc.) on the UNIX file device will be initialized with :UTF-8. Note that the UNIX and DSK file devices reference the same files (although some caution is needed because {UNIX} does not simulate Medley versioning), but the device name in a file name ({UNIX}/Users/... vs. {DSK}/Users/...) selects the particular device. The default setting above applies only to files specified with {UNIX}; a separate default entry for DSK must be established to change its default from :XCCS.

The user can also specify the external format on a per-stream basis by putting a function on the list STREAM-AFTER-OPEN-FNS. After OPENSTREAM opens a stream and just before it is returned to the calling function, the functions on that list are applied in order to arguments STREAM, ACCESS, PARAMETERS. They can examine and/or change the properties of the stream, in particular, by calling STREAMPROP to change the external format from its device default.

Mapping between Unicode and XCCS character codes

The XCCS/Unicode mapping tables are defined by the code-mapping files for particular XCCS character sets. These are typically located in the Library> sister directory

```
..>Unicode>Xerox>
```

and the variable UNICODEDIRECTORIES is initialized with a globally valid reference to that path. The global reference is constructed by prepending the value of the Unix environment-variable "MEDLEYDIR" to the suffix >Unicode>Xerox>.

The mapping files have conventional names of the form XCCS-[charsetnum]=[charsetname].TXT, for example, XCCS-0=LATIN.TXT, XCCS-357=RSYMBOLS4.TXT. The translations used by the external formats are read from these files by the function

(READ-UNICODE-MAPPING FILESPEC NOPRINT NOERROR) [Function]

where FILESPEC can be a list of files, charset octal strings ("0" "357"), or XCCS charset names (LATIN EXTENDED-LATIN). Reading will be silent if NOPRINT, and the process will not abort if an error occurs and NOERROR. The value is a flat list of the mappings for all the character sets, with elements of the form (XCCC-code Unicode-code).

When UNICODE is loaded the mappings for the character sets specified in the variable DEFAULT-XCCS-CHARSETS are installed. This is initialized to

```
(LATIN SYMBOLS1 SYMBOLS2 EXTENDED-LATIN FORMS SYMBOLS3 SYMBOLS4 ACCENTED-
 LATIN GREEK)
```

but DEFAULT-XCCS-CHARSETS can be set to a different collection before UNICODE is loaded.

The internal translation tables used by the external formats are constructed from a list of correspondence pairs by the function

(MAKE-UNICODE-TRANSLATION-TABLES MAPPING [FROM-XCCS-VAR]
[TO-XCCS-VAR]) [Function]

This returns a list of two arrays (XCCS-to-Unicode Unicode-to-XCCS)containing the relevant translation information organized for rapid access. If the optional from/to-variables arguments are provide, they are the names of variables whose top-level values will be set to these arrays, for convenience. For the external formats defined above, these variables are *XCCSTOUNICODE* and *UNICODETOXCCS*.

The macro

`(UNICODE.TRANSLATE CODE TRANSLATION-TABLE)` [Macro]

is used by the external formats to perform the mappings described by the translation-tables.

The following utilities are provided for lower-level manipulation of codes and strings.

`(XTOUCODE XCCSCODE)` -> corresponding Unicode

`(UTOXCODE UNICODE)` -> corresponding XCCS code

`(NUTF8CODEBYTES N)` -> number of bytes in the UTF-8 representation of N.

`(NUTF8STRINGBYTES STRING RAWFLG)` -> number of bytes in the UTF-8 representation of `STRING`, translating XCCS to Unicode unless `RAWFLG`.

`(XTOUSTRING XCCSSTRING RAWFLG)` -> The string of bytes in the UTF-8 representation of the characters in XCCSSTRING (= the bytes in its UTF-8 file encoding).

`(HEXSTRING N WIDTH)` -> the hex string for N, padded to `WIDTH`

The `UNICODE` file also contains a function for writing a mapping file given a list of mapping pairs. The function

`(WRITE-TRANSLATION-TABLE MAPPING [INCLUDEDCHARSETS] [FILE])`

produces one or more mapping files for the mapping-pairs in mapping. If the optional `FILE` argument is provided, then a single file with that name will be produced and contain all the mappings for all the character sets in `MAPPING`. If `FILE` and `INCLUDEDCHARSETS` are not provided, then all of the mappings will again go to a single file with a composite name XCCS-csn1,csn2,csn3.TXT. Each cs may be a single charset number, or a range of adjacent charset numbers. For example, if the mappings contain entries for characters in charset `LATIN`, `SYMBOLS1`, `SYMBOLS2`, and `SYMBOLS3`, the file name will be `XCCS-0,41-43.TXT`.

If `INCLUDEDCHARSETS` is provided, it specifies possibly a subset of the mappings in `MAPPING` for which files should be produced. This provides an implicit subsetting capability.

Finally, if `FILE` is not provided and `INCLUDEDCHARSETS` is `T`, then a separate file will be produced for each of the character sets, essentially a way of splitting a collection of character-set mappings into separate canonically named files (e.g. `XCCS-357=SYMBOLS4.TXT`).

UNIXCHAT

The UNIXChat library module is similar to Chat, but communicates with a C-Shell on your own host rather than with another machine.

Requirements

UNIXChat depends on Chat and UNIXComm.

Installation

Load `UNIXCHAT.LCOM` from the library. We recommend that you also load `VTCHAT.LCOM` from the library to ensure support.

User Interface

Use the following procedure to open a Chat window and activate the terminal emulator.

1. Invoke Chat in one of the following ways:

   a. Choose CHAT from the background menu and type **SHELL** at the Host: prompt in the Prompt Window.

   b. Call the Chat function:

      `(CHAT 'SHELL LOGOPTION INITSTREAM WINDOW)`

      See the CHAT module in this manual for the definition of the Chat function. Use the variable `CHAT.DISPLAYTYPES`, if necessary.

   You should now be talking to a UNIX C-Shell in the Chat window.

2. Set your terminal type:

   ```
   prompt% setenv TERM vt100
   ```

   Note: Alternately, shells started from UNIXChat have the shell variable LDESHELL set. Thus, in your .cshrc file you could have

   ```
   if ($?LDESHELL == 1) then

       setenv TERM vt100
       stty erase ↑H

   endif
   ```

   This makes your Chat C-Shells easier to use.

3. When you've completed a Chat session, close the connection by using CLOSE in the right button window menu or by typing exit to the C-Shell.

Note: You can have several Chat windows open to SHELL at the same time.

[This page intentionally left blank]

UNIXCOMM

UNIXComm starts up a UNIX process on a Sun Workstation. UNIXComm, with two functions, provides the user with a stream interface to the SunOS Bourne shell.

Installation

Load `UNIXCOMM.LCOM` from the library.

User Interface

Two functions allow you to open and close SunOS subprocess streams.

`(IL:CREATE-PROCESS-STREAM` *STRING*`)` [Function]

> Interfaces to the SunOS system function (see Chapter 3 of the *SunOS Reference Manual*). This causes a subprocess running the Bourne shell to be spawned; *STRING* is passed as a command to it. A bidirectional stream is returned with input reading data which the process writes to its standard output (`stdout`) and standard error (`stderr`). Output writes data which the process can read from its standard input (`stdin`). EOFP does not return a meaningful value for those streams; instead, you must change the default EOF mechanism for the end of stream.

`(IL:UNIX-STREAM-CLOSE` *STREAM*`)` [Function]

> Returns the numeric status value of the process created by `IL:CREATE-PROCESS-STREAM` if it has exited; otherwise, kills the process. `IL:CLOSE` can be used instead of `IL:UNIX-STREAM-CLOSE` if the process status value is not of interest.

[This page intentionally left blank]

UNIXPRINT

UnixPrint lets you arrange to have hardcopy sent directly to a PostScript printer via a UNIX print command. You can set your default printing host so that it happens automatically.

Installation

Load UNIXPRINT.DFASL. Customize UNIXPRINTCOMMAND.

Then set the two control variables appropriately, as described below:

`DEFAULTPRINTINGHOST` [Variable]

This is a list of printer names, described in the *Interlisp Reference Manual* (refer to the IRM for a general description). To add a PostScript printer to the list, add an entry in the form `(POSTSCRIPT printername)`. To continue the example above, `DEFAULTPRINTINGHOST` should have a value like the following:

```
(SETQ DEFAULTPRINTINGHOST
        '(—— (POSTSCRIPT daisy) ——))
```

`UnixPrinterName` `[Variable]`

A string or symbol, the name of the UNIX printer to which you want output sent. This should be the name that you would give in the `lpr` command. For example, if you normally print files by entering:

```
lpr -Pdaisy...
```

then you should `(SETQ UnixPrinterName "daisy")`. If you do not normally specify a printer name, set `UnixPrinterName` to `NIL`.

Customization

You can get UNIXPRINT to use `lp` or `lpr` by modifying the function `UnixPrintCommand`. Your site may have a printing program other than `lpr`. For futher information about printing on your system, please refer to your system manual.

(**`UnixPrintCommand`** PRINTER COPIES NAME TMPNAME) [function]

Returns a string that is used by `/bin/sh` in the printing of the postscript code. The arguments are PRINTER, COPIES, NAME and TMPNAME. PRINTER is the name of the printer. COPIES is NIL or a fixp specifying how many copies to print . NAME is the string printed on the banner page of your hardcopy. TMPNAME is the name of the temporary file used to store the postscript code for your job.

A call to `(UnixPrintCommand "daisy" 1 "Erik" "/tmp/foot")` should return something like the string `"/usr/ucb/lpr -Pdaisy -#1 -JErik -r -s /tmp/foot"`.

The source code of the function UnixPrintCommand is supplied with Medley. You are encuraged to write your own versions of this function depending on the site he or she uses. The function `UnixPrintCommand` is included in UNIXPRINT.DFASL. Sources and examples for different versions of `UnixPrintCommand` are included in the file UNIXPRINTCOMMAND.

[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

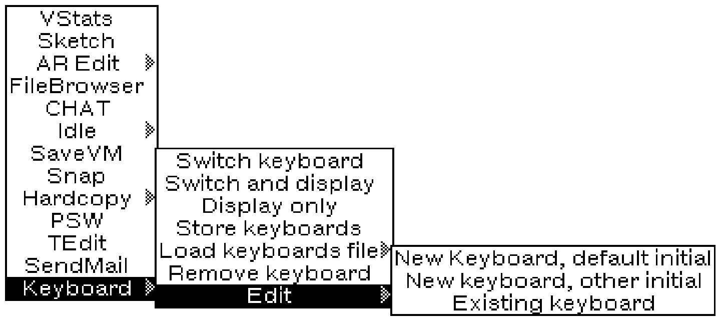
Load `KEYBOARDEDITOR.LCOM` and `VIRTUALKEYBOARDS.LCOM` from the library.

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`.

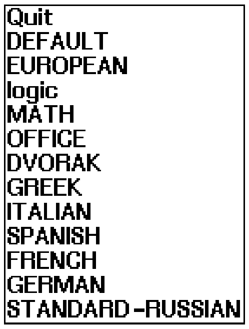


Creating a New Keyboard From a Copy of the Default Keyboard

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 prompts 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 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 are prompted for a name for the new keyboard. The system then displays a menu of the known keyboards from which to choose 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 are not 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 is done on that keyboard and the second argument is ignored.

If *KEYBOARD* is a new name, the editing is done on a copy of *INITIALKEYBOARD*, with *KEYBOARD* as its new name. If *INITIALKEYBOARD* is NIL, the default keyboard is used as a base keyboard.

Examples:

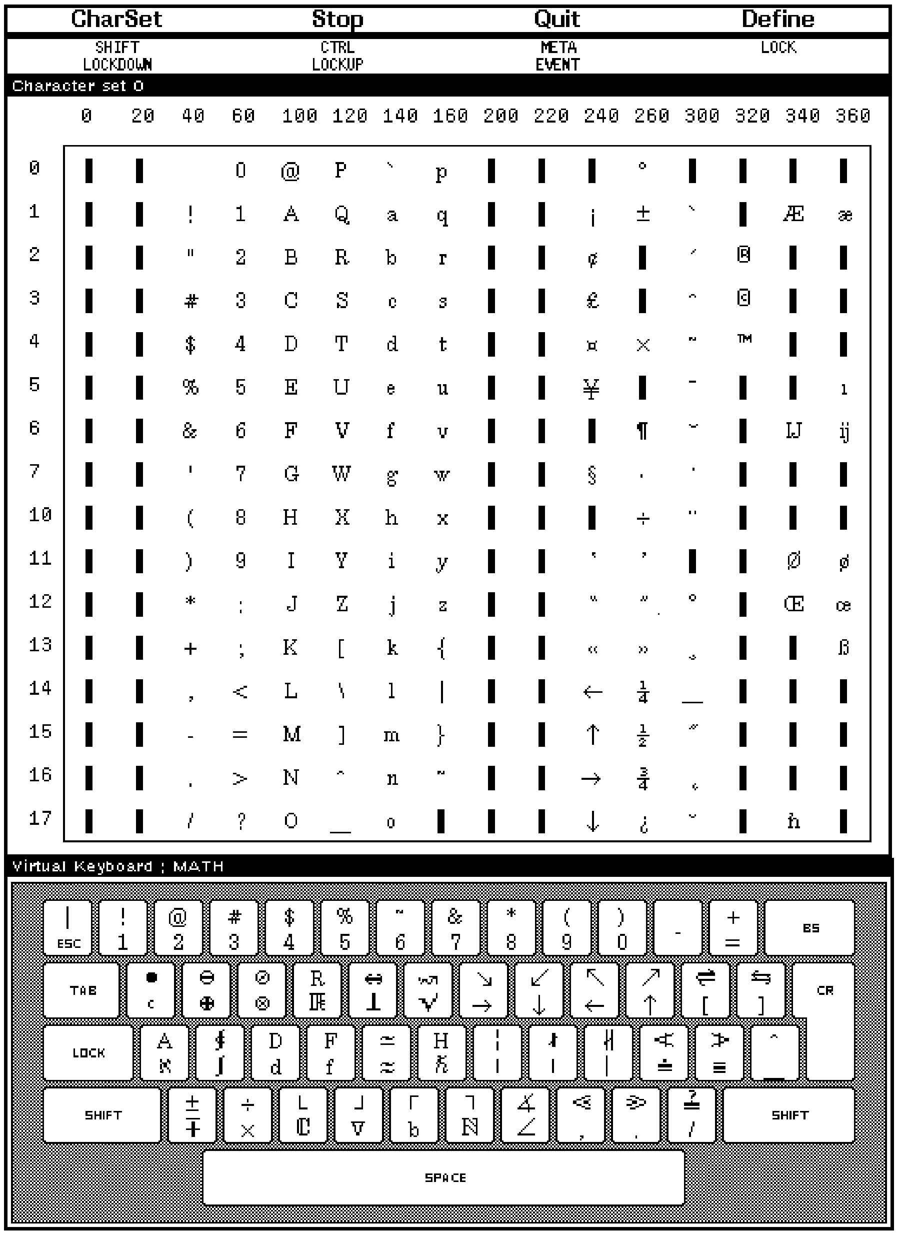
To create a totally new virtual keyboard, call `(EDITKEYBOARD` *NEWNAME*`)`.

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, the command menu is at the top, the character menu in the middle, and the keys menu at the bottom.



	0	20	40	60	100	120	140	160	200	220	240	260	300	320	340	360
0	▌	▌		0	@	P	`	p	▌	▌	▌	°	▌	▌	▌	▌
1	▌	▌	!	1	A	Q	a	q	▌	▌	¡	±	`	▌	Æ	æ
2	▌	▌	"	2	B	R	b	r	▌	▌	¢	▌	´	®	▌	▌
3	▌	▌	#	3	C	S	c	s	▌	▌	£	▌	^	©	▌	▌
4	▌	▌	$	4	D	T	d	t	▌	▌	¤	×	~	™	▌	▌
5	▌	▌	%	5	E	U	e	u	▌	▌	¥	▌	¯	▌	▌	ı
6	▌	▌	&	6	F	V	f	v	▌	▌	▌	¶	˘	▌	Ĳ	ĳ
7	▌	▌	'	7	G	W	g	w	▌	▌	§	·	˙	▌	▌	▌
10	▌	▌	(	8	H	X	h	x	▌	▌	▌	÷	¨	▌	▌	▌
11	▌	▌	)	9	I	Y	i	y	▌	▌	‘	’	▌	▌	Ø	ø
12	▌	▌	*	:	J	Z	j	z	▌	▌	“	”	˚	▌	Œ	œ
13	▌	▌	+	;	K	[	k	{	▌	▌	«	»	¸	▌	▌	ß
14	▌	▌	,	<	L	\	l	|	▌	▌	←	¼	_	▌	▌	▌
15	▌	▌	-	=	M	]	m	}	▌	▌	↑	½	˝	▌	▌	▌
16	▌	▌	.	>	N	^	n	~	▌	▌	→	¾	˛	▌	▌	▌
17	▌	▌	/	?	O	_	o	▌	▌	▌	↓	¿	ˇ	▌	ħ	▌

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 pops 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.

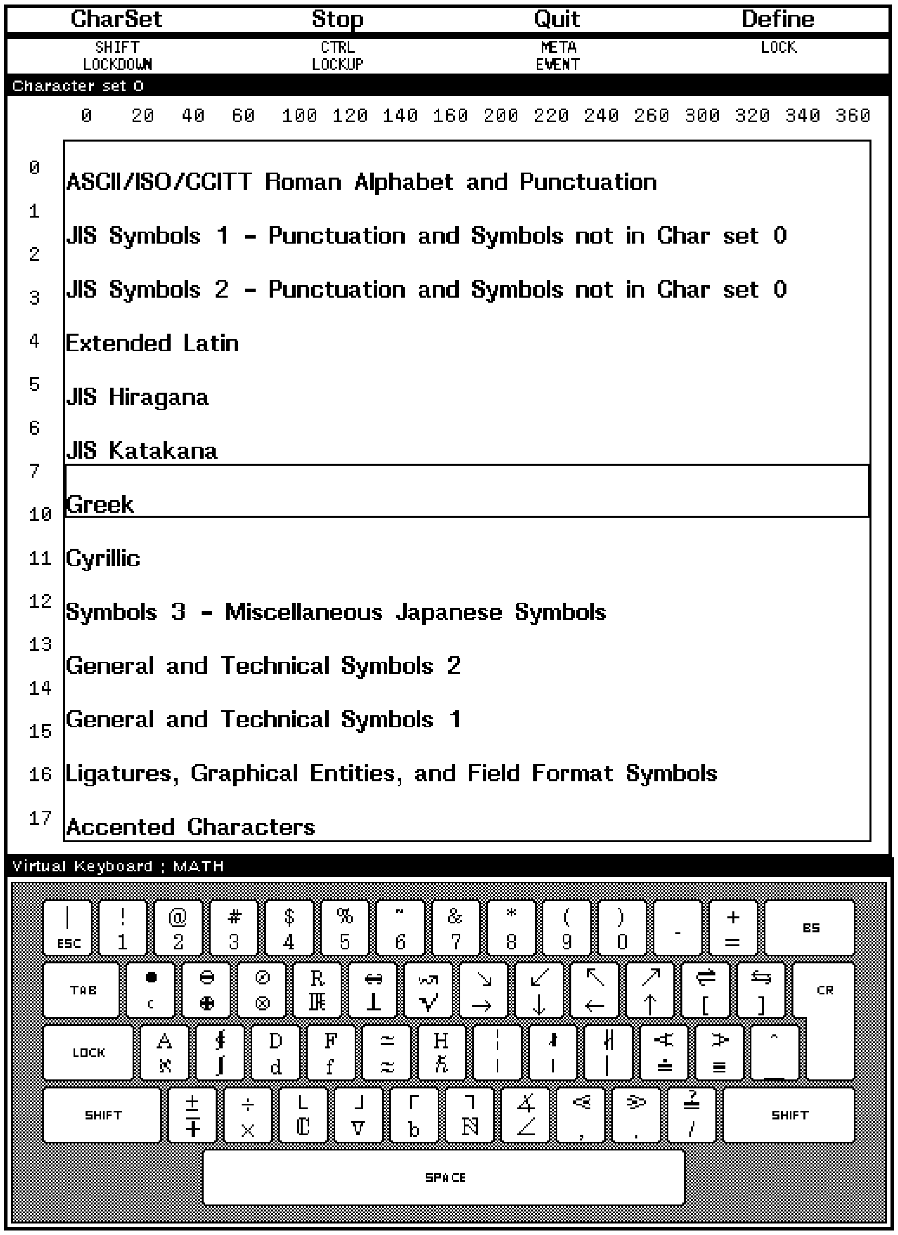

CharSet
Stop
Quit
Define
SHIFT
LOCKDOWN
CTRL
LOCKUP
META
EVENT
LOCK
Character set 0
0 20 40 60 100 120 140 160 200 220 240 260 300 320 340 360
0 ASCII/ISO/CCITT Roman Alphabet and Punctuation
1 JIS Symbols 1 - Punctuation and Symbols not in Char set 0
2 JIS Symbols 2 - Punctuation and Symbols not in Char set 0
3 Extended Latin
4 JIS Hiragana
5 JIS Katakana
6 Greek
7 Cyrillic
10 Symbols 3 - Miscellaneous Japanese Symbols
11 General and Technical Symbols 2
12 General and Technical Symbols 1
13 Ligatures, Graphical Entities, and Field Format Symbols
14 Accented Characters
15
16
17
Virtual Keyboard ; MATH
ESC
BS
TAB
CR
LOCK
SHIFT
SHIFT
SPACE

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 ise 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 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 sends 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 sends the shifted character to the current input stream.

If a key has the LOCKSHIFT property, the lock key is inverted in the keys menu. To change the LOCKSHIFT property of a key, first make the shifted key the current key. 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 adds the newly created keyboard to the list of known keyboards (it will thus appear on future menus). `Quit` exits after modifying the virtual keyboard; `Stop` exits without modifying the keyboard. In both cases the new keyboard is 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]

An Alist of the IDs you use for the keys in the rest of the configuration; i.e., your names for the keys. For simplicity, these are usualy 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 `KEYACTION`s (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 do not 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.

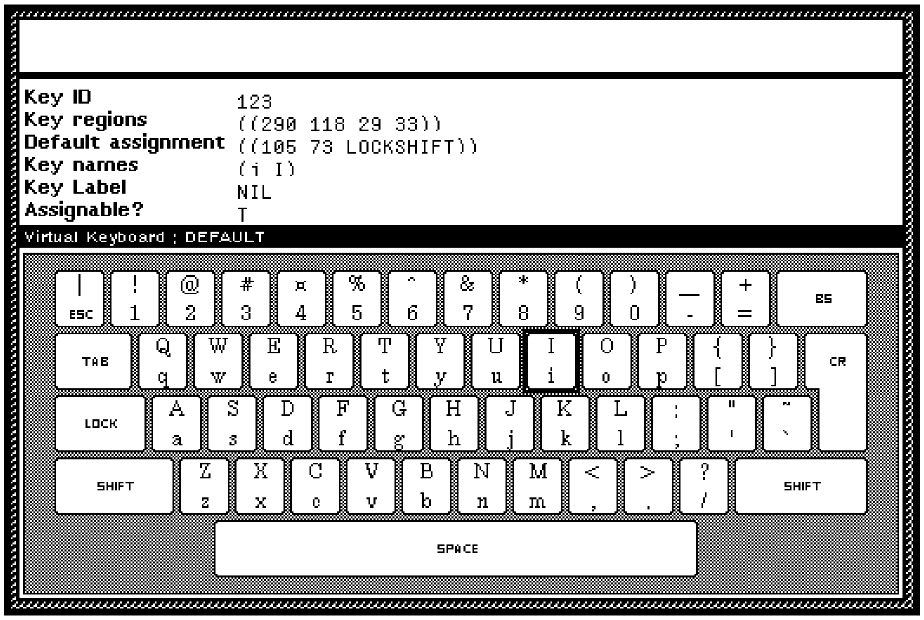
Note: You must save your own configurations. There is no user interface for saving them, nor any automatic scheme.

Editing a Keyboard Configuration

Once you have created a KEYBOARDCONFIGURATION, you can make modest changes to it using the function:

(EDITCONFIGURATION *CONFIGNAME*) [Function]

where C*ONFIGNAME* is the CONFIGURATIONNAME you have assigned to your new configuration. This creates a virtual keyboard editing 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

Virtual Keyboard : DEFAULT

Figure 8. Virtual Keyboard Editing 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 are 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]

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

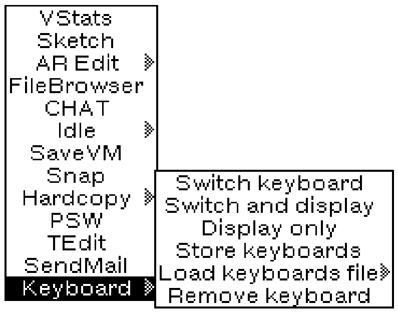
Installation

Load `VIRTUALKEYBOARDS.LCOM` from the library. VirtualKeyboards loads the xxxKEYBOARDS files.

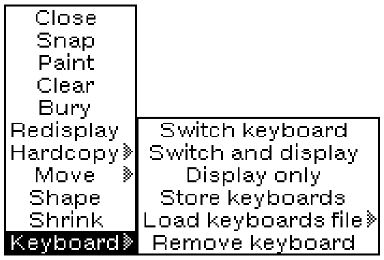
User Interface

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.



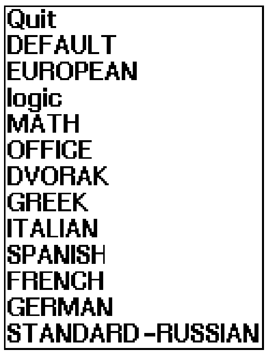
Selecting this item from the default window menu allows you to specify a keyboard for an individual window.



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.

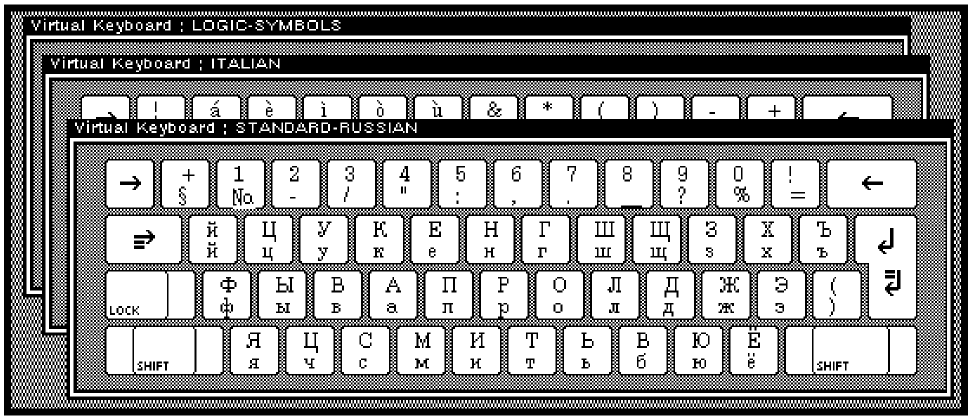

Virtual Keyboard ; LOGIC-SYMBOLS
Virtual Keyboard ; ITALIAN
Virtual Keyboard ; STANDARD-RUSSIAN
LOCK
SHIFT
SHIFT

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.

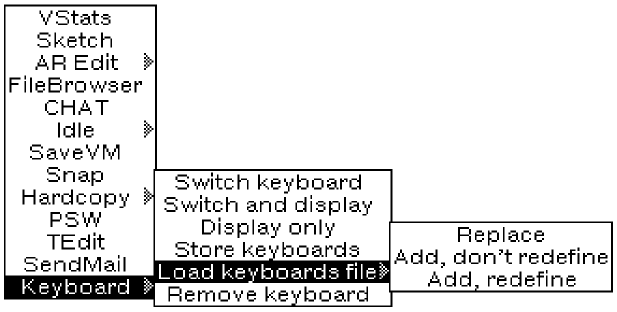
The effect is exactly as if you had pressed a key with that character assigned to it (except that interrupt characters are treated as ordinary characters; i.e., they do not cause an interrupt). The character is sent to the process that has the TTY (usually where the caret is flashing).

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 page intentionally left blank]

WHERE-IS

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. 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` looks 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`also includes 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 semicolon so that only definers on the most recent version are recorded.

If *new* is true, a new database file is 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, all changes 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`prints 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]