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

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

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

Requirements

READNUMBER
SCALEBITMAP

Installation

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

User Interface

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

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

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

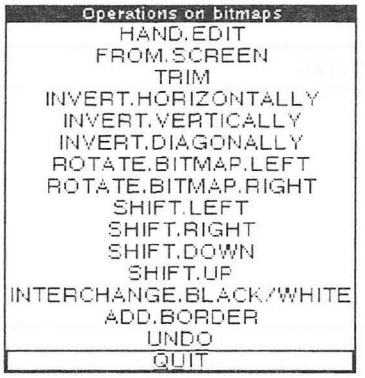
EDIT.BITMAP

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

(EDIT.BITMAP *BITMAP*) [Function]

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

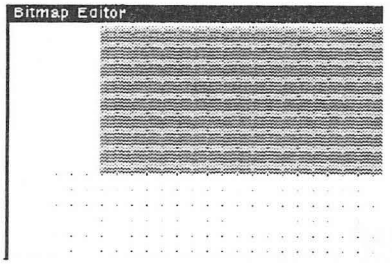
EditBitMap brings up a main menu containing the following items:



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

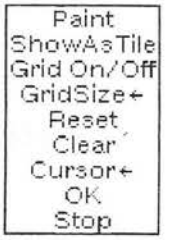
Window

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



Submenu

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

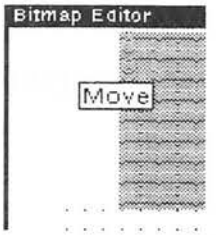


These menu items are described in the system prompt window when an item is selected.

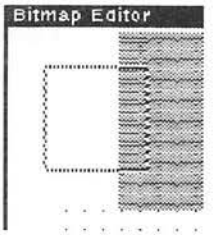
Mouse Buttons

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

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



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



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

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

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

Editing an Existing Bitmap

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

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

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

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

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

Edit the bitmap as needed.

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

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

Viewing an Existing Bitmap

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

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

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

Creating a New Bitmap

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

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

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

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

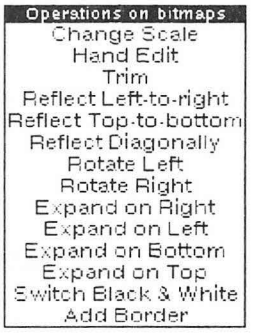
```
(SETQ NEWBITMAPNAME (EDITBM))
```

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

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

Editing a Bitmap in a Document

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



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

Functions

(EDIT.BITMAP *BITMAP*) [Function]

(See above)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(TRIM.BITMAP *BITMAP*) [Function]

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

(FROM.SCREEN.BITMAP NIL) [Function]

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

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

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

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

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

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

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

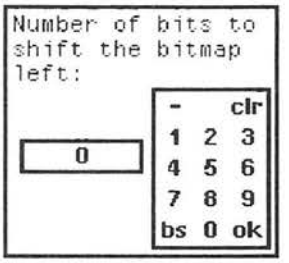
(INTERACT&SHIFT.BITMAP.UP *BITMAP*) [Function]

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

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

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

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



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

Installation

Load ETHERRECORDS from the library.

General Purpose Records

ETHERPACKET [Data type]

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

SYSQUEUE [Data type]

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

QABLEITEM [Record]

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

NS Records

XIP [Record]

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

ERRORXIP [Record]

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

\XIPOVLEN [Constant]

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

\MAX.XIPDATALENGTH [Constant]

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

NSHOSTNUMBER [Record]

A record describing a 48-bit XNS host number.

NSADDRESS [Data type]

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

NSNAME [Data type]

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

PUP Records

PUP [Record]

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

ERRORPUP [Record]

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

PUPADDRESS [Record]

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

\PUPOVLEN [Constant]

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

\MAX.PUPDATALENGTH [Constant]

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

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

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

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

Requirements

TABLEBROWSER

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

Installation

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

User Interface

Starting FileBrowser

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

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

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

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

Next, if you did not specify a pattern by using the FB command, FileBrowser prompts you for a file pattern. Type a pattern, as

described in the section "Specifying What Files to Browse," below.

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

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

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

Specifying What Files to Browse

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

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

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

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

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

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

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

If you leave out the extension of a file name, FileBrowser lists all the files with the specified file name and any extension. If you

omit the extension but include the period that usually precedes the extension, FileBrowser lists only the files with the specified name and *no* extension.

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

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

Examples

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

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

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

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

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

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

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

Using the FileBrowser Window

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

PROMPT window

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

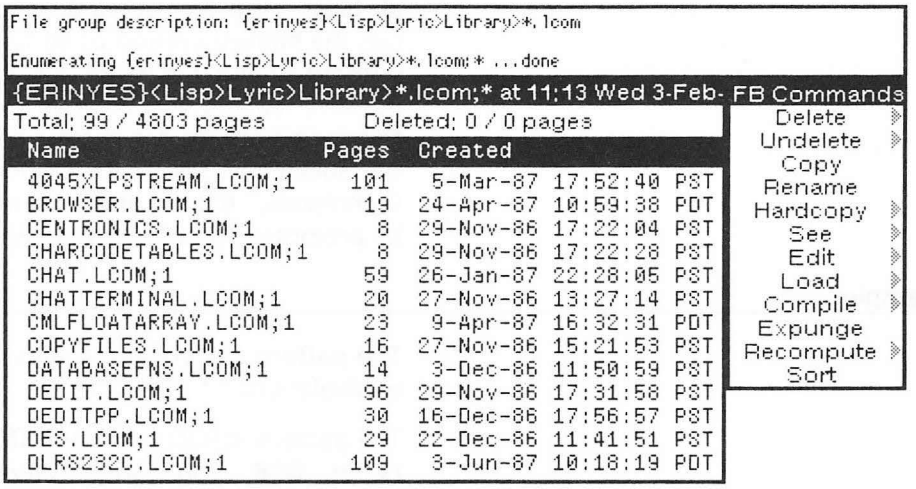
TALLY window

This subwindow immediately below the prompt window keeps a running tally of the total number of files listed in the window and the number of files that you have marked for deletion. In addition, if one of the attributes you are displaying is a size

attribute (Pages or Length, as in the INFO menu, described below), this window maintains a tally of the total number of pages in the files listed and the files marked for deletion.

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

The window is blank while the files are being enumerated.



BROWSER window

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

COMMAND menu

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

INFO menu

An additional subwindow, the Info menu, is not normally displayed. It is used to change the set of file information

(attributes) displayed in the browser (see the section "Getting Information About Files," below).

SCROLL bar

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

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

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

Selecting Files

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

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

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

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

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

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

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

Commands that Require Input

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

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

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

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

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

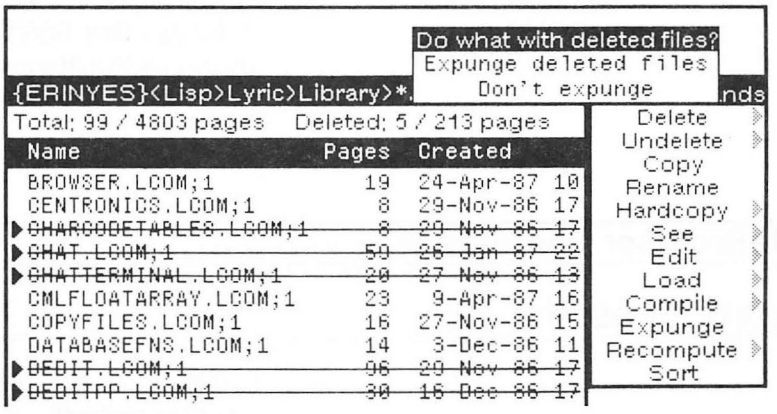
Aborting Commands

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

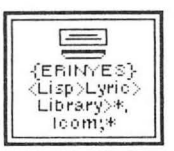
Clicking on the Abort command will immediately abort the current operation. Aborting some commands can take a little while, as FileBrowser may need to do some cleaning up, so the Abort command is greyed out during this time to show you that it is doing something.

Quitting the FileBrowser

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



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



Copy-Selecting Files

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

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

Getting Hardcopy Directory Listings

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

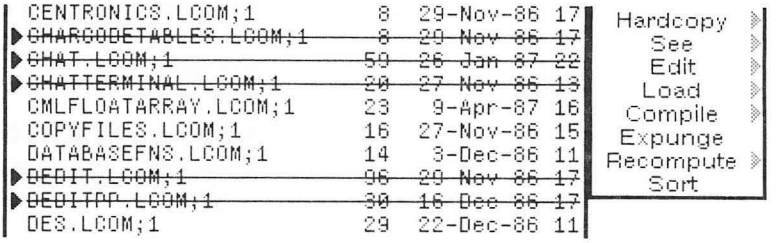
If the browser displays a large number of attributes, or your default printer font is too large, the listing may not accommodate all the attributes on one line, making the listing less readable. You may want to make the listing with fewer attributes, or use a smaller font for the listing (see description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

FileBrowser Commands

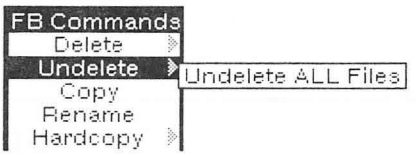
DELETE, UNDELETE

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

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



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.

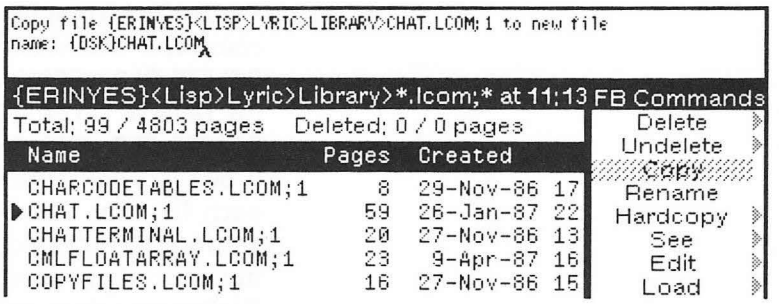
3. Use the COPY command to copy those files.

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.



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.

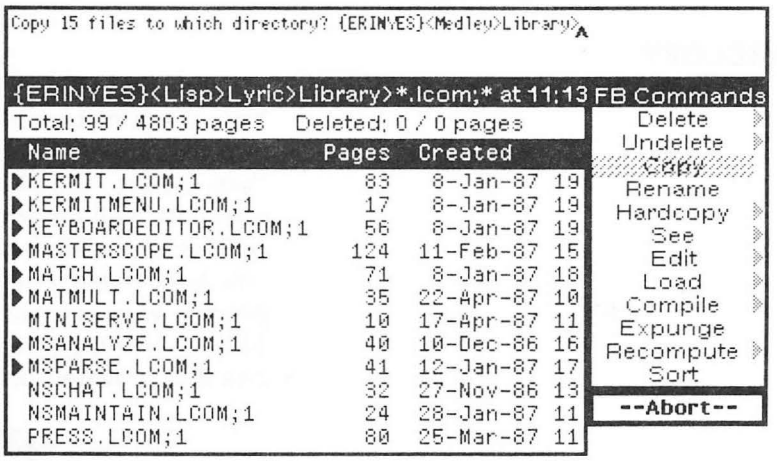
Even files marked for deletion can be copied.

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).



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

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

RENAME

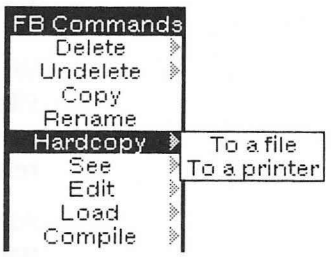
The RENAME command is used for changing the name of a file or group of files, or for moving a file or group of files to a different directory.

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

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

HARDCOPY

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



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

the Interpress print format), then a file of the specified type is made automatically. Otherwise, you are prompted to supply a print format type.

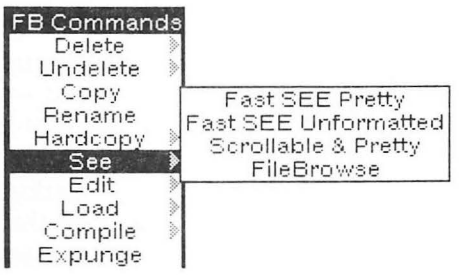
Note: For files stored on servers not supporting random access, FileBrowser is currently unable to determine that a file is in TEdit format unless the file has the extension .TEDIT. Therefore you should use TEdit to hardcopy TEdit files with other extensions. Use FileBrowser's EDIT command (to call TEdit),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

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

SEE

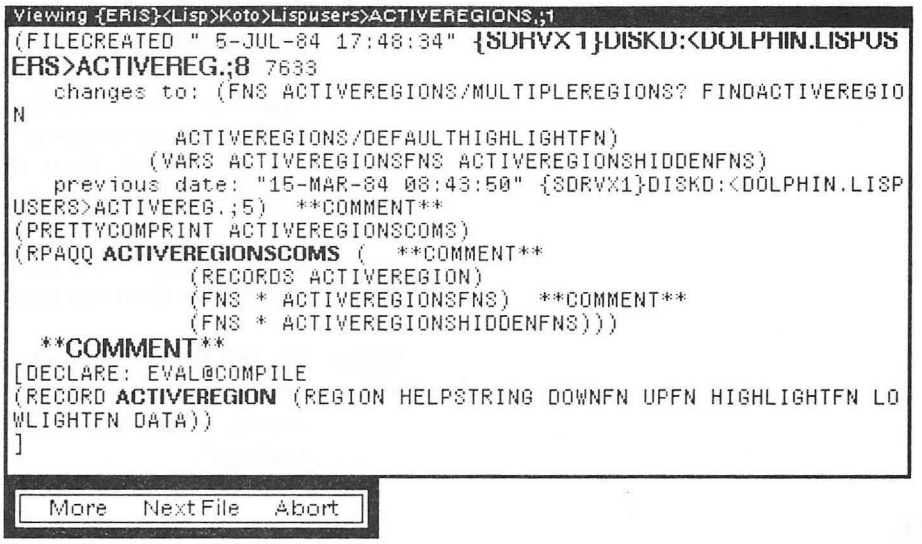
When you browse a directory you sometimes want to see a file before printing or performing some other operation on it. To do this, select the file, then select the SEE command from the command menu. FileBrowser will prompt you to open a window by presenting you with a dashed rectangle and printing a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

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



The two FAST SEE commands display the selected file in the display window one windowfull at a time. When the file fills the window, a small menu appears at the bottom-left corner of the window (or top-left if your display window is at the bottom of the screen) giving you the option of seeing more of the file or

aborting the SEE command. If you issued the SEE command with more than one file selected, you also have the choice of aborting just the display of this file or the entire SEE command.



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).

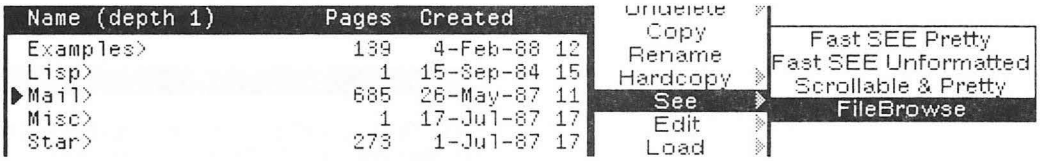
The SEE SCROLLABLE & PRETTY command views a file in a different way. This command brings up a new read-only TEdit

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

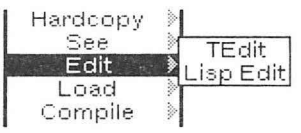
The FILEBROWSE command is a subcommand of SEE used to view a subdirectory in its own FileBrowser window. The selected file must be a (sub)directory. Subdirectory files appear in browsers on XNS file servers when the depth is finite (see the SET DEPTH command), and their names always end in ">".



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.

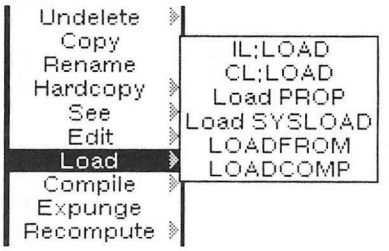
A special display window is opened to give information about the files as they are loaded. When the load is complete, FileBrowser closes the load window.

```
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.

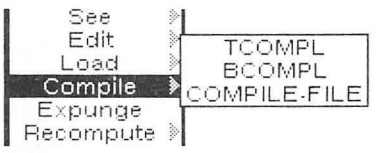
LOADFROM calls the file manager's function LOADFROM. This loads variables and other expressions but not Interlisp functions, and does so in a way that informs the file manager, so that the editor knows where to find the functions.

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.

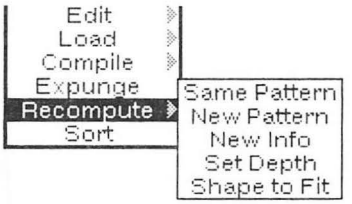
For example, if you open a FileBrowser on your directory, then save several versions of a TEdit file on that directory, the file listing will not display the new versions until you RECOMPUTE.

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".

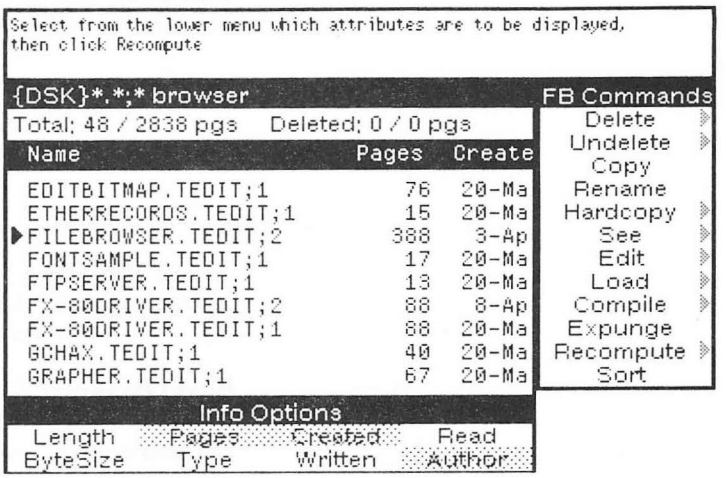
SHAPE TO FIT reshapes the FileBrowser window so that all the attributes in the display are visible at once, eliminating the need to horizontally scroll the window to get at all the information.

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.



The Info Options items have the following meanings:

CREATED The date and time that the content of the file was created. This date changes whenever the file is modified, but does not change when a file is copied or renamed.

WRITTEN The date and time the file was last written to the file system. This date is never older than the Created date, but it can be newer if the file is copied, unmodified, from one file system to another.

READ The date and time the file was last read. This attribute may be blank if the file has never been read.

AUTHOR The login name of the person who wrote the file, or last modified it.

LENGTH The length of the file in (usually eight-bit) bytes.

PAGES The number of 512-byte pages in the file. On some servers, this attribute is blank if the file is empty.

BYTESIZE The size (in bits) of the bytes in the file. In Xerox Lisp this is always eight, but some older computers and file servers allow other sizes.

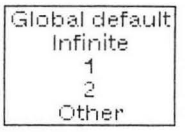
TYPE A value indicating what kind of data the file contains. The usual values of this attribute are TEXT, meaning the file contains just characters, or BINARY, meaning the file contains arbitrary data. Some servers have additional types, such as INTERPRESS for files in Interpress format.

SET DEPTH

XNS file servers support a feature that allows enumerating a directory to a user-specifiable depth. The "depth" of a file reflects the number of subdirectories between it and the root of the enumeration, i.e., the directory or subdirectory you gave in the pattern to FileBrowser, not counting any containing wildcards (asterisks). The immediate descendants of the root are

at depth 1, files in subdirectories of depth 1 are at depth 2, and so on.

Ordinarily, FileBrowser enumerates a directory to the default depth, which is usually unlimited. To enumerate a directory to a different default, use the FB command with argument :DEPTH *n*, for some positive integer *n*, or T for unlimited depth. To change the depth in an existing FileBrowser, use the SET DEPTH command, a subcommand under the RECOMPUTE command. The command offers you a menu of choices:



"Global default" means use the default depth, overriding the depth at which this browser was last enumerated. "Infinite" means use no depth limit (same as depth T). "1" and "2" are common depth choices; to choose some other numeric value, select "Other" and enter the value via the displayed keypad.

The SET DEPTH command does not affect the current display. It takes effect the next time you use the RECOMPUTE or FILEBROWSE commands from the same browser.

During a RECOMPUTE, if a subdirectory appears at the specified maximum depth, its descendants are not enumerated; rather, the subdirectory itself appears as an entry in the browser display. This entry can be selected, just like a file, but only a small number of commands can be used on it: you can RENAME it, you can DELETE it if it has no descendants, and you can FILEBROWSE it. It has attributes, just as ordinary files do. Its page size is the size of the entire subtree rooted at the subdirectory.

Note: Depth currently affects only XNS servers; all other devices ignore it and enumerate to their own default depths. In addition, due to a bug in XNS Services 10, depth is ignored for nontrivial patterns, i.e., anything but "*.*".

SORT

The SORT command allows you to sort the files in the browser by any attribute of the files displayed in the browser. Selecting SORT brings up a menu of attributes by which to sort. This menu includes all the attributes currently displayed in the browser (such as Creation Date, Author), plus the choice Name. For some attributes you can sort forward or backwards; the choice is on a submenu, and the default is generally in the order of numerically greatest (e.g., size) or most recent (e.g., creation date) first.

If the attribute you select is not Name, then the file names displayed in the browser will be reformatted to include their directory portion (if there are any subdirectories below the browser's main pattern), as the subdirectory information is no longer implicit in a file's position in the browser.

The sort order Name, Decreasing Version is the default order in which browsers initially are created.

Customizing FileBrowser and Using the Programmer Interface

FileBrowsers are created programmatically by the function FILEBROWSER, or by type-in from an Executive window with the FB command.

Note: All functions, variables and literals in this section are in the Interlisp package. You'll have to use the prefix IL: if you are using another Executive.

FileBrowser Functions

(FILEBROWSER *FILESPEC ATTRIBUTES OPTIONS*) [Function]

Creates a FileBrowser on files matching the pattern *FILESPEC* and displaying the values of the specified *ATTRIBUTES* for each file. Returns the main window of the browser.

If *ATTRIBUTES* is missing or NIL, it defaults to the value of FB.DEFAULT.INFO (below). See FB.INFO.MENU.ITEMS for the complete set of allowable attributes.

OPTIONS is a list in property-list format. The currently implemented properties and their values are as follows:

:REGION A screen region in which FILEBROWSER will open the browser; if this option is omitted, FileBrowser will prompt you for a region.

:DEPTH The depth to which the enumeration should be performed. Affects only XNS servers (see SET DEPTH).

:TITLE The title for the main browser. If this is omitted, the title derives from *FILESPEC*, and is updated whenever the RECOMPUTE command is used.

:MENU-TITLE The title for the command menu. Defaults to "FB Commands".

:MENU-ITEMS The set of ITEMS composing the command menu. The default is the value of FB.MENU.ITEMS.

FB *FILESPEC ATTR1 . . . ATTRN* [Command]

This is an Executive command for creating FileBrowsers. FB creates a FileBrowser on files matching the pattern *FILESPEC* and displaying the attributes *ATTR1* through *ATTRN*, or the value of FB.DEFAULT.INFO if no attributes are specified. It prompts you for a window region. If a keyword appears in the command line, the remainder of the line from that point on is interpreted as the *OPTIONS* argument to FILEBROWSER.

For example, the Executive command

```
FB *.MCOM LENGTH CREATIONDATE
```

browses all files on the connected directory with extension MCOM, displaying the length in bytes and creation date for each. The command

```
FB * :DEPTH 1
```

browses the connected directory to depth 1, displaying the default attributes.

FB always returns NIL.

FileBrowser Variables

There are several global variables that can be altered to affect FileBrowser's behavior. You can set them by typing

(SETQ *VARIABLENAME NEWVALUE*)

to your Executive window, and can save their values with a VARS command in your initialization file.

FB.DEFAULT.INFO [Variable]

A list specifying which attributes should be displayed for each file. The elements of this list are the Lisp names for the attributes you want displayed. The choices are CREATIONDATE, WRITEDATE, READDATE, LENGTH, SIZE, BYTESIZE, AUTHOR, and TYPE. The attribute SIZE corresponds to the info item "Pages".

For example,

```
(SETQ FB.DEFAULT.INFO '(CREATIONDATE LENGTH))
```

would cause all new FileBrowsers to display exactly the attributes creation date and length in bytes for each file.

FB.INFO.MENU.ITEMS [Variable]

The list of items in the menu used by the NEW INFO command. Each element of the list is of the form (*LABEL ATTRIBUTE "DOCUMENTATION"*). If you add new attributes to this variable, you should also add corresponding entries to FB.INFO.FIELDS.

FB.INFO.FIELDS [Variable]

A list describing, for each attribute, the format in which it is displayed. In addition, the order of attributes in this list is the order in which they are displayed in a browser window. Each element is of the form (*ATTRIBUTE HEADER WIDTH FORMAT PROTOTYPE*), where

ATTRIBUTE is the name of the attribute, which must be a valid attribute for GETFILEINFO;

HEADER is a string displayed in the header line above the main browser window;

WIDTH is the total width in pixels to allocate for printing the values of this attribute, including trailing space;

FORMAT is NIL for ordinary values, DATE for attributes whose value is a date, or (FIX *N*) for integer values;

PROTOTYPE is a string describing the widest value you expect the field to have.

All values are printed left-justified in the allotted space, except for format (FIX *N*), used for integer values, which are right-justified in a field *N* pixels wide, with *WIDTH–N* pixels left for trailing space.

If *PROTOTYPE* is present, then the *WIDTH* and *N* fields for the item are ignored, and the width is taken to be the width of the prototype string in the browser's font (FB.BROWSERFONT), plus two characters' worth of space between columns.

FB.DEFAULT.NAME.WIDTH [Variable]

The amount of space, in points, to use for displaying file names in a browser, initially 140. The name column is automatically expanded if enough names are too wide. You can set this larger if you routinely browse directories of long file names.

FB.ICONFONT [Variable]

The font in which the file pattern is displayed on the browser icon, initially eight-point Helvetica. The value of this variable should be a font descriptor, as returned by FONTCREATE.

For example,

```
(SETQ FB.ICONFONT (FONTCREATE 'MODERN 10 'BOLD))
```

FB.BROWSERFONT [Variable]

The font in which the information in the main display window is printed, initially 10-point Gacha.

FB.PROMPTFONT [Variable]

The font in which prompt messages are printed, initially eight-point Gacha.

FB.PROMPTLINES [Variable]

The number of lines in the prompt window, initially three.

FB.HARDCOPY.FONT [Variable]

Specifies the font to use when producing hardcopy directory listings. Initially NIL, which means to use the font class DEFAULTFONT.

FB.HARDCOPY.DIRECTORY.FONT [Variable]

Specifies the font to use for subdirectory names, if there are any, in hardcopy directory listings. Initially NIL, which means to use the font class ITALICFONT.

FB.DEFAULT.EDITOR [Variable]

Specifies the editor to call by default when the main Edit command is selected. Its value is one of the following:

TEDIT Use the TEdit text editor.

LISP Use the Lisp structure editor, as in the Lisp Edit subcommand.

NIL Use the Lisp structure editor if the selected file is a File Manager Lisp source file, TEdit otherwise.

Other Names the entrypoint function of the editor of choice. The editor is called with a single argument, the file name.

The initial value of FB.DEFAULT.EDITOR is TEDIT.

FILING.ENUMERATION.DEPTH [Variable]

The system variable that controls the default depth to which a directory is enumerated. The value is a positive integer, or T for unlimited depth. The initial value is T.

Adding FileBrowser Commands

You can add your own commands to FileBrowser by adding items to FB.MENU.ITEMS and writing functions to handle the commands. This section describes the format of menu commands and a set of functions that are useful for the implementation of FileBrowser commands.

FB.MENU.ITEMS [Variable]

A list of the items that appear on FileBrowser's command menu. You can add new FileBrowser commands by adding new elements to the end of this list. After your change, any new FileBrowser will have the added commands.

Each element of FB.MENU.ITEMS is of the form (*LABEL YOURFN "EXPLANATION"*), where

LABEL is the name of the command, as it is to appear in the menu;

YOURFN is the name of the function to be called when the command is selected, and

EXPLANATION is the explanation to be printed when the mouse cursor is held over the command.

While *YOURFN* is Executing, the menu command is grayed out, and FileBrowser is "locked" so that no other commands or processes can access it.

You can have subcommands as well if you make the menu command be of the form (*LABEL YOURFN "EXPLANATION"* (*SUBITEMS item1. . . itemN*)), where each *itemJ* is recursively of the same form as a menu command.

FileBrowser calls *YOURFN* with four arguments: (*YOURFN BROWSER KEY ITEM MENU*), as follows:

BROWSER is FileBrowser object in control of this browser window.

KEY is the mouse key pressed (left or middle).

ITEM is the menu item that was selected.

MENU is the command menu.

YOURFN can also be a list of two elements, (*FN ARG*), where *ARG* is an arbitrary value that is passed, unevaluated, as the fifth argument to *FN*. This is useful for writing one function that implements several subcommands that are similar to the original command.

Any additions to FB.MENU.ITEMS should be saved with the file manager command APPENDVARS, so that the new items are added to the end of the menu, and your changes will not interfere with any changes to built-in FileBrowser commands in new FileBrowser releases.

(FB.TABLEBROWSER *BROWSER*) [Function]

Returns the TABLEBROWSER object belonging to FileBrowser *BROWSER*. See the documentation for the module TABLEBROWSER for further operations you might perform on one of these browsers.

(FB.SELECTEDFILES *BROWSER NOERRORFLG*) [Function]

Returns a list of table items representing the files currently selected in *BROWSER*. If there are no selected files, this prints

No files are selected

in the prompt window, unless *NOERRORFLG* is true, in which case this function quietly returns NIL.

(FB.FETCHFILENAME *ITEM*) [Function]

Returns the full name of the file denoted by *ITEM*, one of the table items returned by FB.SELECTEDFILES.

(FB.PROMPTWPRINT *BROWSER X1...XN*) [Function]

Prints the strings *X1* through *XN* in *BROWSER*'s prompt window. The item T is printed as a carriage return (i.e., a command to go to a new line).

(FB.PROMPTW.FORMAT *BROWSER FORMAT-STRING* &REST *ARGS*) [Function]

Prints to *BROWSER*'s prompt window by applying the Common Lisp function FORMAT to *FORMAT-STRING* and *ARGS*.

(FB.PROMPTFORINPUT *PROMPT DEFAULT BROWSER ABORTFLG DONTCLEAR*) [Function]

Prompts for your input in *BROWSER*'s prompt window. *PROMPT* is the prompt string, *DEFAULT* is the default answer. Returns your input as a string, or NIL if there is no input, or if it was aborted with control-E. If there is no input and *ABORTFLG* is true, prints "...aborted". The prompt window is first cleared (as at the beginning of a command), unless *DONTCLEAR* is true.

(FB.ALLOW.ABORT *BROWSER*) [Function]

Enables the Abort button on *BROWSER*. This should be called from the function implementing any FileBrowser command of indefinite duration.

Troubleshooting Problems with FileBrowser

When FileBrowser returns the message "No files in group *FILENAMEPATTERN*" when you know those files exist, the file server is probably down or rejecting connections. If this is so, your only option is to wait until the server is functioning again, and then give the Recompute command. In the case of an NS file server, the enumeration of files can also fail if you do not have sufficient access privileges; this condition is usually noted by a message in the system prompt window.

When you try to expunge a file and FileBrowser displays the message "Can't expunge FILENAME," it may be because you don't have write access to the file, or someone else is reading the file. However, the most common reason is that the file is still open. Be sure to close any TEdit windows in which you may still be viewing the file. If you have recently issued a HARDCOPY command for the file, a background process may still be working on the file. If that's not the problem, you can get a list of open streams by typing (OPENP) at the prompt in an Interlisp Executive window. If one of them is open on the file you want to delete, you can close it by passing the stream to the function CLOSEF. To close all open streams (not recommended unless you're sure it is okay), you can type (MAPCAR (OPENP) 'CLOSEF). If you don't find an open stream, and the server is a Leaf or XNS file server, you may have a disagreement between Lisp and the server on what files are open, something that can occur if you had aborted an OPENSTREAM operation. Call (BREAKCONNECTION "servername") to reset the connection, then try again.

[This page intentionally left blank]

FONTSAMPLE

FontSample provides a set of tools for generating Interpress masters for a font sampler, and is intended to allow you to see what results on a printer when you specify a font. This is useful because there may be several font substitutions between when you specify a font (for example in TEdit) and when a printer actually renders the font.

The set of font mappings is a function of the local font substitutions in a particular workstation, the workstation's environment (which fonts are available to it at that time, if the font file server is temporarily unavailable), which printer is being used, and which font files are currently installed on the printer.

For example, Lisp might substitute Terminal for Gacha, and the printer might substitute Modern for Terminal. Thus you could request Gacha and get Modern. The font sampler is intended to display the final result of these substitutions.

Requirements

FONTSHEETSx.IP (where x = 1, ... 7)

Installation

Load FONTSAMPLE.LCOM from the library.

User Interface

To find the cumulative effect of all font substitions at a given time, environment, and printer, you should generate the Interpress masters (which reflect the environmental and Lisp mappings) and then send them to various printers (which reflect the printer mappings).

```
(SEND.FILE.TO.PRINTER 'FONTSHEET1.IP)
(SEND.FILE.TO.PRINTER 'FONTSHEET2.IP)
etc.
```

Short sample masters for all currently supported Interpress fonts can be found in the Lisp Library with the names FONTSHEET1.IP, FONTSHEET2.IP ... FONTSHEET7.IP. These reflect the default Lisp font mappings.

Note: These environmental mappings may change from release to release, so new masters should be printed for every release.

Functions

(FNT.MAKEBOOK *OUTFROOTNAME LISTOFFONTS PRINTFN PERPAGE)*

[Function]

This function enumerates fonts across multiple output files. Multiple files are necessary because some printers cannot handle documents with many fonts. The function iterates over the fonts in *LISTOFFONTS*, invoking *PRINTFN* (which has arguments as in FNT.DISPLOOK) *PERPAGE* times per output file. The files are named *OUTFROOTNAME* with the page number concatenated at the end. If *LISTOFFONTS* is the atom ALL, then FONTSAVAILABLE is invoked for all Interpress fonts (this is an extremely slow operation). *PRINTFN* defaults to FNT.DISPLOOK; *PERPAGE* defaults to 18. Each font in *LISTOFFONTS* should be a FONTLIST.

(FNT.DISPTBLE *STREAM FONT*) [Function]

This function prints a description of the font and characters 0 to 254 of the font specified by *FONT* on *STREAM.* It expects a letter-sized output stream. If used with FNT.MAKEBOOK, *PERPAGE* should be 1.

(FNT.DISPLOOK *STREAM FONT*) [Function]

This function prints a description of the font and representative characters of *FONT* on *STREAM*. If used with FNT.MAKEBOOK, *PERPAGE* should be 18.

Limitations

The font sheets are applicable only to Interpress printers.

Example

```
(SETQ FONTLISTLIST (LIST '(MODERN 10 MRR 0 INTERPRESS)
'(CLASSIC  8 BRR 0 INTERPRESS)))

(FNT.MAKEBOOK 'FOO FONTLISTLIST 'FNT.DISPTBLE 1)
```

generates two files named FOO1 and FOO2 each containing output from one invocation of FNT.DISPTBLE. Thus FOO1 has a font table for (MODERN 10 MRR 0 INTERPRESS), FOO2 has a font table for (CLASSIC 8 BRR 0 INTERPRESS).

FTPSERVER

FTPServer implements a simple PUP FTP server protocol for a Xerox workstation. The server is typically run as a background process on one machine to allow other machines remote access to the files on its disk.

Requirements

Ethernet connection to a remote host.

Installation

Load FTPSERVER .LCOM from the library.

Functions

(FTPSERVER *FTPDEBUGLOG*) [Function]

Creates a process named FTPSERVER that listens on the standard PUPFTP server socket for incoming connection requests. When one arrives, FTPSERVER services it, then returns to its listening state. The process continues to run until killed.

If *FTPDEBUGLOG* is non-NIL, it should be an open file/stream to which tracing information is printed during the life of the process.

If *FTPDEBUGLOG* is T, output goes to a newly created window. *FTPDEBUGLOG* can also be a REGION, specifying where the window is to be created.

FTPSERVER.DEFAULT.HOST [Variable]

Initially DSK. This is the default host for files requested of the server via FTP. Setting this to FLOPPY, for example, would serve files off the machine's floppy drive.

Note: FTPSERVER.DEFAULT.HOST can also be set to the name of a remote host, but this has limited utility, as it doesn't handle passwords correctly.

Limitations

The current implementation is a simple tool which allows file transfer between Xerox machines and supports only one remote

connection at a time. Because of this, files cannot be loaded indirectly, i.e., via the filecoms of another file.

For example, suppose FOO loads BAR which loads WOO. When FOO is being loaded, it will attempt to load BAR. But FTPServer cannot support the second connection required to load BAR while the first connection is still open to load FOO. (This is similar to the case of trying to load FOO and BAR when they are on different floppies.)

Therefore, you should load files in an order that prevents recursive loads: in this example, load WOO, then BAR, then FOO.

Delete (DELFILE) operation is now supported. Rename (RENAMEFILE) operation is not implemented. FTPServer is best suited for simple COPYFILE operations.

Examples

An alternative way of specifying the host from the remote machine is to make the host name be the "device" field of the file name specification.

For example, if machine M is running FTPServer, another machine could ask for directory of {M}FLOPPY:FOO.* to get a listing of M's {FLOPPY}FOO*.

To address your host, you may use the results of ETHERHOSTNAME. If on your host (ETHERHOSTNAME NIL T) evaluates to 123#456#, then on a remote machine you can access file FOO on the host by:

```
{123#456#}FOO
```

FX-80Driver prints text and graphics on Epson FX-80-compatible printers. It implements a full device-independent graphics interface for the FX-80, and can print source code, TEdit documents, bitmaps and windows at a variety of qualities and speeds.

The FX-80Driver contains two printer drivers: a fast driver, for quick printing of draft-quality text, and a high-quality driver, for slower printing of mixed-font text and graphics. You can print early revisions of a document in fast mode, and then switch to high-quality mode for the final copy. The matrix shown in Figure 1 illustrates the capabilities of each mode:

	Fast	High-quality
TEdit	monofont only	yes
Sketch		yes
Windows		yes
Lisp source code	monofont only	yes
Grapher		yes

Figure 1. FX-80 printer drivers

For historical reasons, FX-80 in this document refers to any and all of the Epson FX-80 family of dot-matrix graphics printers. The module supports the FX-80, FX-85, FX-86 and FX-286. The Epson printers vary in speed and carriage width, but share a common command language.

Requirements

RS232 or TTY cable (see the wiring diagrams in the Introduction of this manual).

Serial interface card in the printer.

DLRS23C or DLTTY.

Installation

FX-80 Serial Interface

The FX-80Driver module requires that your Epson be equipped with a suitable serial interface (such as the Hanzon Universal Card).

The interface should be set up with XOn/XOff flow control enabled, 9600 baud or slower, 1 stop bit, 8 bit characters, no parity.

(See *The Hanzon Universal Card* booklet for instructions on the DIP switch settings.)

FX-80 DIP Switch Settings

The FX-80 should have its DIP switches set as shown in Figure 2.

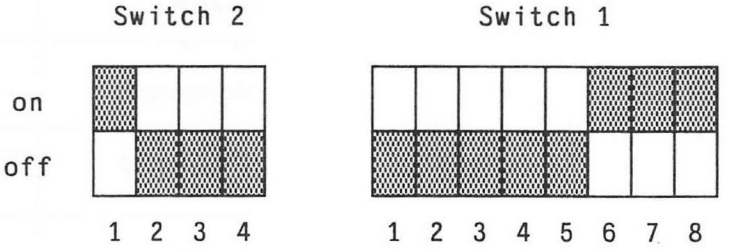


Figure 2. FX-80 DIP switch settings

Switch 1 says no automatic linefeed, no automatic paper feed, no buzz on paper-out, and to allow no software deactivation of the printer.

Switch 2 says to use the USA character set, Pica type, allocate 2KB for user-defined characters, allow paper-out detection, and print zeros as zeros.

Note: For the FX-85, -86 and -286 DIP switch settings, consult the corresponding *Epson User's Manual*.

Software

Load FX-80DRIVER.LCOM and the required .LCOM modules from the library.

Store all of the font files (file names ending with .displayfont) corresponding to the fonts you wish to use on some convenient directory or directories. HQFX80-FONT-DIRECTORIES should be a list that contains these directories; it should be the same as DISPLAYFONTDIRECTORIES.

Set FASTFX80-DEFAULT-DESTINATION (determines where output to the FASTFX80 lineprinter device goes) and HQFX80-DEFAULT-DESTINATION (determines where output to the HQFX80 lineprinter device goes) to one of the following values; they need not be the same:

Destination	RS232 port	TTY port	file
Value	{RS232}	{TTY}	FileName
Speed	9600 max.	4800 max.	n/a

Load the appropriate device driver for each of these destinations: DLTTY.LCOM for the TTY port, and DLRS232C.LCOM for the RS232C port.

Run the function RS232C.INIT or TTY.INIT (as appropriate), and set the baud rate to match the setting on the printer.

User Interface

You can set up the FX-80 to be your default printer, send FX-80 output to a file for later printing, or programmatically open an image stream that produces output on the FX-80.

Having the FX-80 set up as your default printer means that you can print the contents of windows by selecting the HARDCOPY menu item on the window of interest. You can also use the HARDCOPY - TO A FILE submenu item to spool your output for later printing. And you can write programs that use the OPENIMAGESTREAM function to create FX-80 format graphics output.

Printing in Fast Mode

You can print in fast mode by sending output to the printer FASTFX80 or by opening an image stream to a file with extension FASTFX80. This mode is called fast because it uses the printer's built-in font, which allows a tight encoding of the document to be printed. Fidelity to the original document is not as good as in high-quality mode.

The following restrictions apply:

Special characters (that is, most Xerox Network Systems extended characters, such as the bullet or dagger; see CharCodeTables, VirtualKeyboards in this manual) are ignored.

Only one font is supported (though roman, italic, and bold typefaces do work).

Graphics (lines, underlines, bitmaps) are ignored.

Multiple column output does work.

Set FX-80 Fast Mode

To set your default printer to be a fast mode FX-80, make the list (FASTFX80 FASTFX80) the CAR of the list DEFAULTPRINTINGHOST.

Set FX-80 Destination

To set the default destination of all output to {LPT}.fastfx80, set the variable FASTFX80-DEFAULT-DESTINATION to an appropriate file name string. See the table above; the file name could be that of a regular file like {DSK}SPOOLED-FAST-OUTPUT.

Set FX-80 Page Size

To set the driver's page size to match the paper in the printer, set the two variables \FASTFX80.INCHES-PER-PAGE (page height in inches) and \FASTFX80.INCHES-PER-LINE (page width in inches) to appropriate values. The defaults are 11 and 8.5, respectively. These can be set in your Lisp INIT file.

Print a File

Select the HARDCOPY command from the background (right-button) menu. The system first formats the file for printing. Then, when the FX-80Driver actually starts transmitting to the printer, a small abort window, bearing the name of the document and the name of the printer, will appear near the top of your screen.

Abort a Print Job

Clicking on the item marked ABORT in the print window will cleanly terminate the printing of the document.

Note: After aborting a print job, you may need to turn the printer off and back on to make sure that other files will print successfully.

Printing in High-Quality Mode

Print in high-quality mode by sending output to the printer HQFX80, or by opening an image stream on a file with type HQFX80. High-quality mode printing supports all of Xerox Lisp's device-independent graphics operations, as well as multi-font printing and the XNS extended character set. It prints at 72 dot-per-inch resolution. Fidelity to the original document is better than in fast mode, though printing speed is slower.

Set HQ Mode

To set your default printer to be a high-quality FX-80, make the list (HQFX80 HQFX80) the CAR of the list

DEFAULTPRINTINGHOST. You can use the PRINTERMENU module or your favorite structure editor to do this.

Set Destination

To set the default destination of all output to {LPT}.hqfx80, set the variable HQFX80-DEFAULT-DESTINATION to an appropriate file namestring. This could be "{TTY}", "{RS232}", or even the name of a regular file like "{DSK}spooled-hq-output".

Set Page Size

To set the driver's page size to match the paper in the printer, set the two variables \HQFX80.INCHES-PER-PAGE (page height in inches) and \HQFX80.INCHES-PER-LINE (page width in inches) to appropriate values. The defaults are 11 and 8.5, respectively. These can be set in your Lisp INIT file.

Print a File

Select the HARDCOPY command. The system first formats the file for printing. Then, when the FX-80Driver actually starts transmitting to the printer, a small abort window, bearing the name of the document and the name of the printer, will appear near the top of your screen.

Note: After printing a document on HQFX80, you may need to turn the printer off and back on before you can print with FASTFX80 on that printer.

Abort a Print Job

See above.

FX Printer Compatibility

(FX80-PRINT *&KEY THING-TO-PRINT LANDSCAPE? COMPRESS? HIGH-QUALITY?*) [Function]

THING-TO-PRINT may be one of a window, bitmap, or file path name. If *THING-TO-PRINT* is a path name, the file will be treated as either a TEdit or Lisp source file, and printed in the appropriate style.

In the window or bitmap cases, *LANDSCAPE?* specifies landscape printing (X-coordinates run down the left margin) when non-NIL;

COMPRESS? specifies FX-80 compressed printing mode.

If *HIGH-QUALITY?* is non-NIL and *THING-TO-PRINT* is a path name, output will be sent to the default high-quality FX-80 printer, otherwise to the default fast FX-80 printer.

The *LANDSCAPE?*, *COMPRESS?*, and *HIGH-QUALITY?* arguments all default to NIL.

Limitations

Landscape printing has not been implemented.

Examples

Send text output to fast FX-80:

```
(SETQ FX-80 (OPENIMAGESTREAM "{LPT}.FASTFX80"))
(CL:FORMAT FX-80 "HELLO, WORLD~%%")
(CL:CLOSE FX-80)
```

Print source code on fast FX-80 (assuming the FastFX80 is not your default printer, but is on the list DEFAULTPRINTINGHOST):

```
(LISTFILES (HOST FASTFX80) "{DSK}MYPROGRAM")
```

Note: Source code is stored in pre-pretty-printed form on the file. The pretty-printer's default linelength (width of a line in characters) is normally 102, which is too wide for the FastFX-80s 8.5-inch wide page. To create source files which print nicely on the fast FX-80, you should set the variable FILELINELENGTH to a more appropriate value before you MAKEFILE. 70 works nicely on 8.5-inch paper with a standard font profile, though your mileage may vary.

Print source code in the the fast FX-80 mode, assuming the FastFX80 is your default printer:

```
(LISTFILES "{DSK}MYPROGRAM")
```

Print TEdit file in fast FX-80 mode, assuming the FastFX80 is your default printer:

```
(LISTFILES "{WAYCOOL:}<PUBLIC>GENSYM.TEDIT")
```

Print text and graphics in high-quality mode:

```
(SETQ FX-80 (OPENIMAGESTREAM "{LPT}" 'HQFX80))
(MOVETO 300 300 FX-80)
(CL:FORMAT FX-80 "HELLO, WORLD~%%")
(DRAWCIRCLE 300 300 230 '(ROUND 8) NIL FX-80)
(CL:CLOSE FX-80)
```

Print source code in high-quality mode, assuming the high-quality FX-80 is not your default printer, but is on the list DEFAULTPRINTINGHOST:

```
(LISTFILES (HOST HQFX80) "{DSK}MYPROGRAM")
```

Note: See the previous note regarding FILELINELENGTH and the fast FX-80. The same holds for high-quality FX-80 printing, and we recommend 70 as the value for FILELINELENGTH.

Print source code in high-quality mode, assuming the high-quality FX-80 is your default printer:

```
(LISTFILES "{DSK}MYPROGRAM")
```

Print TEdit file in high-quality mode, assuming the high-quality FX-80 is your default printer:

```
(LISTFILES "{WAYGNARLY:}<PUBLIC>MAGNUMOPUS.TEDIT")
```

[This page intentionally left blank]

GCHax contains functions that are useful for tracking down storage leaks, i.e., objects that should be garbage but do not get garbage collected. There are functions for examining reference counts, locating pointers to objects, and finding circularities (which are among the chief culprits in storage leaks).

Typically, you might turn to GCHax when you notice that STORAGE claims there are more instances of a data type in use than you believe there should be.

Installation

Load GCHAX.LCOM from the library.

Functions

Storage

The function STORAGE displays statistics on the amounts and distribution of the virtual memory space that has been allocated. If you suspect your program may have storage leaks (e.g., because (VMEMSIZE) keeps growing without obvious reason), this function is the place to start to get an indication of which kinds of objects are not being reclaimed. STORAGE is part of the standard Lisp sysout; you need not have loaded GCHAX to use it.

(STORAGE *TYPES PAGE-THRESHOLD IN-USE-THRESHOLD*) [Function]

With no arguments, STORAGE displays statistics for all data types, along with some summary information about the space remaining. The optional arguments let you refine the display.

If *TYPES* is given, STORAGE only lists statistics for those types. *TYPES* should be a type name or list of type names.

If *PAGE-THRESHOLD* is given, then STORAGE omits types that have fewer than *PAGE-THRESHOLD* pages allocated to them. The default *PAGE-THRESHOLD* is 2, so types that are not currently in use (consume no storage) do not appear unless you specify a *PAGE-THRESHOLD* of zero.

If *IN-USE-THRESHOLD* is given, then STORAGE omits types that have fewer than *IN-USE-THRESHOLD* instances in use (allocated and not yet freed).

For example, (STORAGE '(ARRAYP BITMAP)) lists only statistics for the types ARRAYP and BITMAP; (STORAGE NIL 6) lists only statistics for data types that have at least six pages allocated. (STORAGE NIL NIL 100) lists only statistics for data types that have at least 100 instances still in use.

The STORAGE function displays, for each Lisp data type, the amount of space allocated to the data type, and how much is currently in use. The display looks something like this:

```
Type          Assigned          Free items     In use    Total alloc
              pages [items]
FIXP             66    8448        7115         1333         447038
FLOATP           24    3072        2412          660         734877
LISTP          2574 ~298584        5294       ~293290        3545071
ARRAYP            8     512         245          267          48199
   . . .
```

Type — Is the name of the data type, as given to DATATYPE or the Common Lisp DEFSTRUCT.

Assigned — Is how much of your virtual memory is set aside for items of this type. Memory is allocated in quanta of two pages (1024 bytes). The numbers under Assigned show the number of pages and the total number of items that fit on those pages. The tilde (~) on the LISTP line indicates that the number is approximate, since cdr-coding makes the precise counting of lists impossible—the amount of memory consumed by any particular list cell varies depending on its contents and how it was allocated.

Free items — Shows how many of the assigned items are available to be allocated (by the Interlisp *create* or the Common Lisp *make-* constructs); these constitute the free list for that data type.

In Use — Shows how many items of this type are currently in use, i.e., have pointers to them and hence have not been garbage collected. If this number is higher than your program seems to warrant, you may want to look for storage leaks. The sum of Free items and In Use is always the same as the total Assigned items.

Total Alloc — Is the total number of items of this type that you have ever allocated (created), or at least since the last call to BOXCOUNT that reset the counter.

STORAGE also prints some summary information about how much space is allocated and available collectively for fixed-length items (mainly data types, both user and built-in), variable-length items (arrays, bit maps, strings), and symbols. The variable-length items have fixed-length headers, which is why they also appear in the printout of fixed-length items. For example, the line printed for the data type BITMAP says how many bit maps have been allocated, but the figure displayed as "assigned pages" counts only the headers, not the space used by the variable-length part of the bitmap. The variable length portion is accounted in the summary statistics for "ArrayBlocks", where it is lumped with all other users of variable-length space, as it is not possible for the system to more finely discriminate the users of the space.

```
Data Spaces Summary
                                  Allocated          Remaining
                                    Pages              Pages
Datatypes (incl. LISTP etc.)         4370              \
ArrayBlocks (variable)               5770              -- 47758
ArrayBlocks (chunked)                2626              /
Symbols                              1000                  1048
```

```
variable-datum free list:
le 4              18 items;        72 cells.
le 16             84 items;       865 cells.
le 64             38 items;      1019 cells.
le 256            76 items;      7580 cells.
le 1024            2 items;      1548 cells.
le 4096           11 items;     18568 cells.
le 16384           1 items;      4864 cells.
others             2 items;     59565 cells.

Total cells free:    94081  total pages:  736
```

In the summary, Remaining Pages indicates how many more pages are available to be allocated to each type of datum. There is a single figure for both fixed- and variable-length objects, because they are allocated out of the same pool of storage.

Variable-length objects are allocated in two different ways, reflected in the items "variable" and "chunked." The distribution of the former among several different sized free lists is shown next.

Storage Leak Tracking Functions

The functions in GCHax are oriented toward finding leaks that involve items of some data type not getting garbage collected.

There are two main kinds of leaks:

> Items that are unintentionally being held onto.
>
> Items that no user structure is pointing to but are not collected because of the nature of the garbage collector.

Examples of the former are structures assigned to global variables and left there after the program finishes.

Examples of the latter are principally circular structures — structures where you can follow a chain of pointers from an object that eventually returns to the same object. Circular lists, such as you get from (NCONC A A), are a special case of circular structures. See comments in "Limitations" below.

Note: All functions listed below have names beginning with \ to remind you that you are dealing with system internals, and to proceed with at least a little caution. Although these functions are generally safe, in that their casual use will not cause arbitrary damage, you certainly can produce unintended side effects.

In particular, the functions \SHOWGC and \COLLECTINUSE have modes in which they return a list of some kind of pointer; beware of unintentionally holding on to such a list (e.g., by having it get onto the history list), thereby preventing the eventual garbage collection of any of those pointers.

Useful for keeping values off the history list are the Executive command SHH for completely inhibiting history list entry, and the idiom (PROG1 NIL operation), e.g., (PROG1 NIL (INSPECT value)) to inspect a structure without holding on to a pointer to

the inspect window. You may find it convenient to define your own Exec command to do inspection, e.g.,

```
(DEFCOMMAND IN (OBJ TYPE)
    (PROG1 NIL (INSPECT OBJ TYPE)))
```

The reference counts of all objects in the system are maintained in a global hash table, called the GC reference count table. Some or all of its contents can be viewed with the following function:

(\SHOWGC *ONLYTYPES COLLECT FILE CARLVL CDRLVL MINCNT*) [Function]

Displays on *FILE* (default T) all objects in the GC reference count table whose reference count is at least *MINCNT*, whose default value is 2.

If *ONLYTYPES* is given, it is a list of data type names to which \SHOWGC confines itself.

If *COLLECT* is T, \SHOWGC returns a list of all the objects it displays.

CARLVL and *CDRLVL* are print levels affecting the displaying of lists; they default to two and six, respectively. In the listing, collision entries in the reference count table are tagged with a *. Reference count operations on pointers in collision entries are much slower than on noncollision entries.

Objects with reference count of one (1) do not appear explicitly in the reference count table, so cannot be viewed with \SHOWGC, even if you set *MINCNT* as low as 1.

Note that if *COLLECT* is T, then the reference count of all the collected items is now one greater, due to the pointer to each from the returned list.

(\REFCNT *PTR*) [Function]

Returns the current reference count of *PTR*. Pointers that are not reference counted (e.g., symbols and small integers) are considered to have reference count 1. Since pointers from the stack (e.g., PROG variables) do not affect reference counts, it is possible for the reference count of an object to be zero without the object being garbage collected.

Note: If you call \REFCNT from the Common Lisp interpreter, e.g., by typing it at top-level, the answer is almost always too large by 1, as the interpreter itself holds reference-counted pointers to the arguments to the function it is calling. The same problem besets \FINDPOINTER (below). The problem does not exist from the Old Interlisp Exec, which uses the Interlisp interpreter. You can also avoid the problem by explicitly invoking the Interlisp interpreter; e.g.,

(EVAL '(\REFCNT *expression*)).

(\#COLLISIONS) [Function]

Returns a list of four elements:

Number of entries in the reference-count table, i.e., the number of objects in memory whose reference count is not 1;

Number of entries that are in collision chains;

Ratio of these numbers, i.e., the fraction of all entries that are in collision chains;

Ratio of the number of entries to the size of the hash table.

(\#OVERFLOWS) [Function]

Returns a list of four elements like \#COLLISIONS, but instead counts only objects whose reference count has overflowed (is greater than 62). Reference count operations on such objects are significantly slower than on other objects.

(\COLLECTINUSE *TYPE PRED)* [Function]

Is useful when (STORAGE *TYPE*) shows more objects in use than you think is right, but you can't find any such pointers yourself.

TYPE is a data type name or number other than LISTP. \COLLECTINUSE returns a list of all objects of that type that are thought to be in use, i.e., not free.

If *PRED* is supplied, it is a function of one argument. \COLLECTINUSE returns only objects for which *PRED* returns true. *PRED* must not allocate storage; you probably want it to be a compiled function.

Note: \COLLECTINUSE should be used with care. In a correctly functioning system, \COLLECTINUSE is generally safe. However, if the free list of *TYPE* has been smashed so that some free objects are not on it, this function can make matters much more confused, especially if the first 32-bit field of the data type in question contains a pointer field.

(\FINDPOINTER *PTR COLLECT/INSPECT? ALLFLG MARGIN ALLBACKFLG*)

[Function]

Provides a brute-force approach to answering the question, "Who has a pointer to *x*?" \FINDPOINTER searches virtual memory, looking for places where *PTR* is stored. The search is not completely blind: unless *ALLFLG* is true, it does not look in places that cannot have reference-counted pointers, such as pname space or the stack. However, if the reference count of the object is zero, \FINDPOINTER searches the stack (and only the stack, if *ALLFLG* is NIL), since in this case there is no hope of finding pointers in the usual reference-counted spaces. If *ALLFLG* = :STACK, then \FINDPOINTER searches the stack in addition to places that contain reference-counted pointers, but not other unlikely places.

\FINDPOINTER prints out a description of each place *PTR* is found. If it is found in a list, it asks whether to recursively search for pointers to the list, so you can track lists back to a more identifying place, such as a symbol value cell or some data type. It recurses without asking if *ALLBACKFLG* is true. If *PTR* is found in a typed object, \FINDPOINTER names the field, if the data type declaration is available, and asks if you want to recursively search for pointers to this object. In either case, the search stops once enough places have been found to account for *PTR*'s reference count (unless *ALLFLG* is T).

If *COLLECT/INSPECT?* is true, \FINDPOINTER saves the identifiable pointers in a list. If *COLLECT/INSPECT?* = COLLECT, the list of pointers is returned as value; otherwise, it is offered for inspection.

MARGIN is the left margin (in units of characters) by which the reports of locations are initially indented. The default is zero. Recursive searches for pointers are indented relative to this position.

The current version does not know how to parse array space, so if *PTR* is found in an array, the best it can do is print the memory address where it found it, usually something of the form {}#nn,nnnnn. In addition, \FINDPOINTER doesn't even try to find *PTR* as a literal inside a compiled code object, since such references are not cell-aligned. Thus, \FINDPOINTER is really most helpful if the pointer is stored in fixed-length data space (e.g., in a field of a data type, or as the top-level value of a symbol); fortunately, this handles most of the interesting cases in practice.

Note: Of course, since it touches (potentially) a huge percentage of your virtual memory, \FINDPOINTER is completely disruptive of your working set.

(\FINDPOINTERS.OF.TYPE *TYPE FILTER*) [Function]

Calls \FINDPOINTER on each pointer in use of type *TYPE* that satisfies *FILTER*, a function of one argument, the pointer. A *FILTER* of NIL is considered the true predicate. *FILTER* can also be a list form to evaluate in which the variable PTR is used to refer to the pointer in question.

\FINDPOINTERS.OF.TYPE is essentially the same as

```
(for PTR in (\COLLECTINUSE TYPE)
      when <FILTER is satisfied>
      do (\FINDPOINTER PTR))
```

except that it takes care to discard the cells of the list returned from \COLLECTINUSE before calling \FINDPOINTER, to avoid seeing one extra reference per object.

For example,

```
(\FINDPOINTERS.OF.TYPE 'STREAM '(NOT (OPENP PTR)))
```

searches for pointers to all streams that are not currently open.

(\SHOW.CLOSED.WINDOWS) [Function]

Collects all windows that are not currently open or icons of open windows, then opens each window one by one.

For each window, you are prompted to press the left mouse button to close the window and go on to the next, or press right to do something different. In the latter case, you are prompted again to press the left button if you would like to search for pointers to the window, using \FINDPOINTER, or press the right button to just leave the window open on the screen and proceed.

Returns the total number of windows examined.

(\SHOWCIRCULARITY *OBJECT MAXLEVEL*) [Function]

Follows pointers from *OBJECT*. If it finds a path back to itself, it prints that path. This function is not exceptionally fast, and deliberately (for performance reasons) does not detect circularities in lists; it simply bottoms out on lists at *MAXLEVEL*, which defaults to 1,000. Circular lists are usually obvious enough anyway.

(\MAPGC *MAPFN INCLUDEZEROCNT*) [Function]

Maps over all entries in the GC reference count table, applying *MAPFN* to three arguments: the pointer, its reference count (an integer), and *COLLISIONP*, a flag that is T if the entry is a collision entry. Entries with reference count zero are not included unless *INCLUDEZEROCNT* is T. This function underlies \SHOWGC. Some care is required in the writing of *MAPFN*; it should try to minimize any reference-counting activity of its own, and in particular avoid anything that would decrement the reference count of the pointer passed to it.

Limitations

GCHax is not very useful for finding ordinary circular lists, as the typical system has vast amounts of list structure, with nothing to distinguish the interesting ones.

However, if the circular list also contains instances of user data types, then those data types will tend to show up as overallocated, and hence amenable to the search functions in this module.

\FINDPOINTER does not know how to locate pointer arrays of more than 64 elements, so it is not helpful if a pointer you seek is located only in such an array.

[This page intentionally left blank]