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

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

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

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

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

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

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

Requirements

Hardware

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

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

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

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

4045 PROM and Software Compatibility

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

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

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

Software

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

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

Installation

Software

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

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

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

4045 Emulation Mode Selection

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

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

On = 630
Off = 2700

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

4045 Port Selection

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

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

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

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

4045 Port Initialization

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

R RS232

T TTY

C Centronics

S Server (in which the 4045 is connected to another 1108/86 and the current machine is printing to it via Ethernet), and

D The default port. The software then automatically loads the correct communications module for the port selected.

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

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

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

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

Using the 4045 as a Default Printing Host

Printing Source or TEdit Files

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

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

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

FILE is a Source or Tedit file.

HOST is '.4045XLP.

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

EXAMPLES:

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

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

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

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

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

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

Printing Windows

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

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

Creating 4045XLP Master Files

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

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

```
(SETQ FOO (OPENIMAGESTREAM 'MYMASTERFILE.4045XLP))
(BITBLT MYBITMAP 0 0 FOO 0 0)
(CLOSEF FOO)
```

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

Printing a master file is easy. Evaluate

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

and it is printed without the delay for formatting.

Printing via FTPserver

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

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

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

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

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

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

Example:

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

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

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

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

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

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

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

Changing Modes for International 4045s

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

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

This function changes the internal parameters of the software to allow printing on A4 paper with the international fonts. Mode is a string, either "USA" or "INTERNATIONAL", with the default being "USA". Do not use this function unless you have the international font set and A4 paper tray on a non-USA 4045. A4

page size is 2475 pixels wide by 3525 pixels high in portrait, and 3525 x 2475 in landscape mode.

4045XLPStream Options

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

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

4045 Parameter Names and Values

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

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

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

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

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

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

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

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

PRINTHEADER — Either a string, T or NIL; default is NIL. Controls the printing of a header page describing the job being printed. T indicates print

the page, NIL indicates not to print the page. If it is a string, it indicates to use this string as the title of the header page.

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

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

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

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

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

Set Parameters

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

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

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

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

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

Get Parameters

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

Will return the value of any of the parameters listed.

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

For example,

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

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

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

Get, Set Parameters via Inspector Window

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

Examples

4045 image streams are created by the OPENIMAGESTREAM function.

Opening a 4045 Stream

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Using a 4045XLP Stream

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

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

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

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

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

```
(DSPNEWPAGE 4045STREAM)
```

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

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

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

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

```
(CLOSEF 4045STREAM)
```

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

Resetting 4045XLPStream

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

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

Limitations

4045 Fonts

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

Multiple Streams

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

Printing Speed

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

Scale factors

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

Notecards

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

TEdit

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

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

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

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

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

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

Do the actual hardcopy by evaluating

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

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

```
(CLOSEF MYSTREAM).
```

SingleFileIndex

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

Sketch

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

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

[This page intentionally left blank]

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

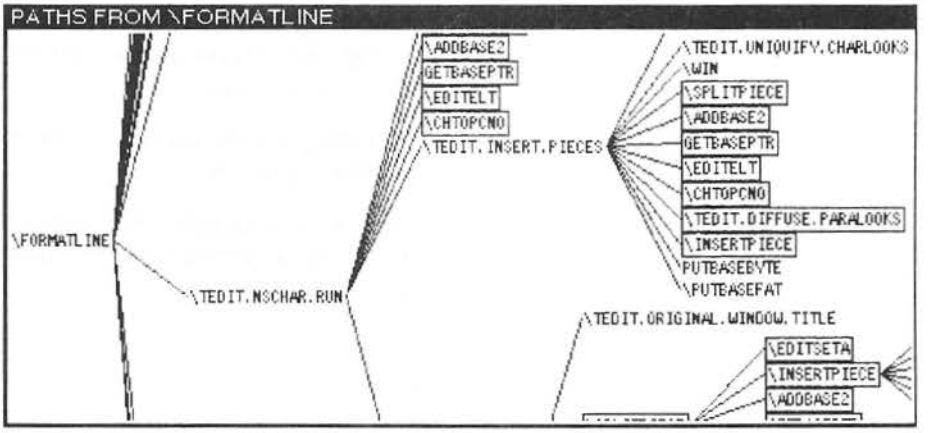
Browser makes it easier to use the MasterScope SHOW PATHS command by making it easier to read its results. This is how SHOW PATHS looks before Browser is loaded:

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

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

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

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



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

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

Requirements

MASTERSCOPE, GRAPHER

Installation

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

User Interface

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

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

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

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

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

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

Functions

(BROWSER T) turns the Browser on.

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

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

SHOWGRAPH displays the graph.

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

Limitations

Browser works only with MasterScope.

Examples

Type the following into an Interlisp Exec window:

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

[This page intentionally left blank]

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

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

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

Installation

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

Functions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Implementation Notes

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

Limitations

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

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

Requirements

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

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

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

Installation

Load CENTRONICS.LCOM from the library.

User Interface

Functions

(CENTRONICS.RESET) [Function]

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

Opening a Centronics Stream

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

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

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

Device Errors

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

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

Limitations

The port is available on Xerox 1109 workstations only.

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

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

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

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

Xerox Character Codes

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

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

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

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

Code assignments are described in detail in:

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

Requirements

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

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

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

Installation

Load CHARCODETABLES.LCOM from the library.

Functions

(SHOWCSETLIST *CSETS FONT*) [Function]

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

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

(SHOWCSETRANGE *FIRSTCSET LASTCSET FONT*) [Function]

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

(SHOWCOMMONCSETS *FONT*) [Function]

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

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

(SHOWCSET *FONT*) [Function]

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

Limitations

This module works only with Interpress fonts.

Examples

```
(SHOWCSETLIST '(0 238 239) '(MODERN10))
```

or

```
(SHOWCSETLIST '(0 #0356 #0357) '(MODERN10))
```

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

```
(SHOWCSETRANGE 24 26 '(MODERN 10 ITALIC))
```

or

```
(SHOWCSETRANGE #030 #032 '(MODERN 10 ITALIC))
```

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

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

[This page intentionally left blank]

CHAT

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

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

There are currently terminal emulators for the following terminals:

Datamedia 2500

DEC VT100

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

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

PUP Chat,

NS Chat (using the GAP protocol),

TCP (ARPANET) TELNET,

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

Each of these is available by loading the corresponding module.

Requirements

DMCHAT

CHATTERMINAL

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

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

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

Installation

Load CHAT.LCOM from the library.

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

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

User Interface

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

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

Opening a Chat Connection

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

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

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

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

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

Login Always perform a login.

Attach Always perform an attach (this is likely to be useful only when opening Chat connections to hosts running the Tops-20 or Tenex operating systems). This will fail if you do not have exactly one detached job.

None Do not attempt to log in or attach.

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

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

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

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

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

Chat Menu

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Customizing Chat

CHAT.DISPLAYTYPES [Variable]

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

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

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

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

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

CHAT.KEYACTIONS [Variable]

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

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

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

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

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

CHAT.INTERRUPTS [Variable]

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

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

CHAT.ALLHOSTS [Variable]

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

CLOSECHATWINDOWFLG [Variable]

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

DEFAULTCHATHOST [Variable]

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

CHAT.FONT [Variable]

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

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

CHAT.WINDOW.SIZE [Variable]

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

CHAT.WINDOW.REGION [Variable]

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

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

CHAT.TTY.PROCESS [Variable]

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

CHAT.EMACSCOMMANDS [Variable]

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

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

CHAT.IN.EMACS? [Variable]

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

CHAT.PROTOCOLTYPES [Variable]

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

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

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

Network Protocols

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

PUP Chat

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

NS Chat

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

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

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

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

Remote System Administration

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

Remote System Executive

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

Interactive Terminal Service

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

External Communication Service

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

ECS hosts typically support a variety of modem connection characteristics (specific combinations of parity, character length, baud rate, and flow control settings). Each connection type is

known by a different Chat host name; check with your system administrator to determine the Chat host name you should use to connect to a particular external host.

TCP Chat

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

RS232 Chat

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

Terminal Emulators

DM2500 Chat

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

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

VT100 Chat

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

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

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

Dual-width and/or dual-height characters

Graphics character set

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

TEdit Chat

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

command to copy blocks of characters from the Chat window to another TEdit window, a Lisp Executive, etc.

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

(<HOSTNAME> <TERMINALTYPENUMBER> TEDIT).

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

[This page intentionally left blank]

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

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

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

Requirements

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

Installation

Load CMLFLOATARRAY.LCOM from the library.

Functions

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

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

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

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

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

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

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

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

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

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

- (MINUS) Negates each element of the array argument.

ABS Computes the absolute value of each element of the array argument.

TRUNCATE The single array argument must be of element-type single-float, but the result array may be any element-type which will accomodate the integer results. Truncates (converts to integer, rounding towards zero) each element of the array argument.

FLOAT The single array argument must be of element-type (unsigned-byte 16), and the result array may be of element-type single-float.

Converts each element of the array argument to a single precision floating point number.

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

+ (PLUS) Computes the element-wise (element by element) sum of the two arguments.

- (MINUS) Computes the element-wise difference of the two arguments.

* (TIMES) Computes the element-wise product of the two arguments.

/ (QUOTIENT) Computes the element-wise quotient of the two arguments.

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

REDUCE-ARRAY is similar to the sequence function REDUCE but is generalized for arrays of arbitrary dimensionality; that is, the

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

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

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

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

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

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

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

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

Limitations

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

[This page intentionally left blank]

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

Installation

Load COPYFILES.LCOM from the library.

Function

(COPYFILES *SOURCE DESTINATION OPTIONS*) [Function]

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

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

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

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

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

OPTION: Conversation Mode

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

QUIET Don't print anything while working.

(OUTPUT *LISTFILE*) Print the name of each file that gets copied on *LISTFILE*. (OUTPUT T) is the default.

TERSE Only print a period (.) for each file moved/copied.

OPTION: Query Mode

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

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

(ASK N) Ask, with default to No after DWIMWAIT seconds.

(ASK Y) Ask, with default to Yes after DWIMWAIT seconds.

OPTION: Version Control

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

RENAME or MOVE Use RENAMEFILE instead of COPYFILE; i.e., the source is deleted afterwards.

ALLVERSIONS Copy all versions and preserve version numbers.

REPLACE If a file by the same name exists on the destination, overwrite it (don't create a new version).

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

OPTION: When To Copy

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

ALWAYS Always copy the file.

> Copy only when a file by the same name but an earlier creation date exists on the destination.

>A Similar to >, but also copy if the file doesn't exist on the destination; i.e., > ALWAYS.

Copy only when a file by the same name but a different creation date exists on the destination.

#A Similar to #, but also copy if the file doesn't exist on the destination, i.e., # ALWAYS.

=A Copy only if there isn't a file of the same name on the destination.

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

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

OPTION: Clean-Up After Copying Files

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

PURGE This involves a separate pass (afterwards): any file on the destination which doesn't have a counterpart on the source is deleted.

PURGESOURCE Converse of PURGE (and used by it): if the file is on the source and not on the destination, delete it.

Limitations

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

Examples

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

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

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

will use RENAMEFILE instead.

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

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

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

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

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

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

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

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

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

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

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

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

For example,

```
(LOAD 'FILE1)
```

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

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

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

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

```
(MAKEFILE 'FILE1)
```

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

Requirements

MASTERSCOPE

Installation

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

User Interface

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

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

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

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

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

Functions

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

LOADDBFLG [Variable]

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

Yes Always try to load the data base when a file is loaded.

No Never try to load the data base when a file is loaded.

Ask (default) Ask whether to load the data base when a file is loaded.

SAVEDBFLG [Variable]

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

Yes Always try to save the data base when a file is saved.

No Never try to save the data base when a file is saved.

Ask (default) Ask whether to save the data base when a file is saved.

(DUMPDB *FILE PROPFLG*) [Function]

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

(LOADDB *FILE ASKFLG*) [Function]

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

Data base files include the date and full file name of the file to which they correspond. LOADDB will print out a warning

message if it loads a data base that does not correspond to the in-core version of the file, and will ask you if you approve.

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

[This page intentionally left blank]

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

Description

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

History

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

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

DEdit

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

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

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

Requirements

DEDITPP

Installation

Load DEDIT.LCOM from the library.

Loading DEdit makes it the default Lisp structure editor.

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

User Interface

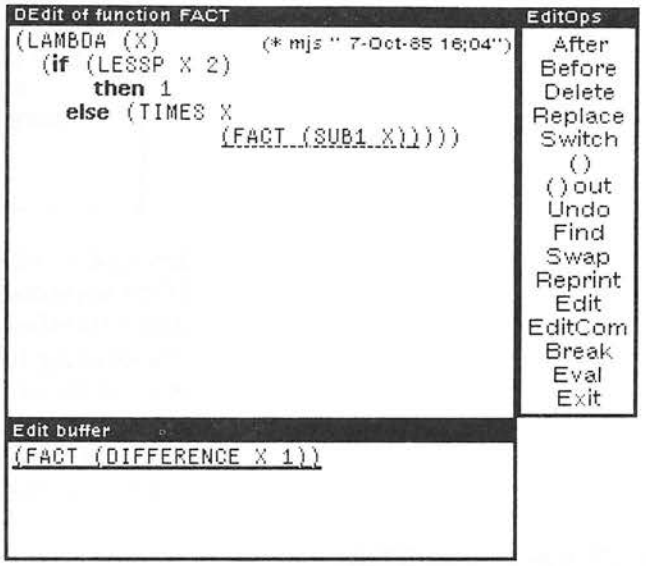
DEdit is normally called using one of the DEdit functions. See also "Advanced Features" below.

DEdit Window

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

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

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



Selecting Objects and Lists

Selection in a DEdit window is as follows:

The left button selects the object being directly pointed at.

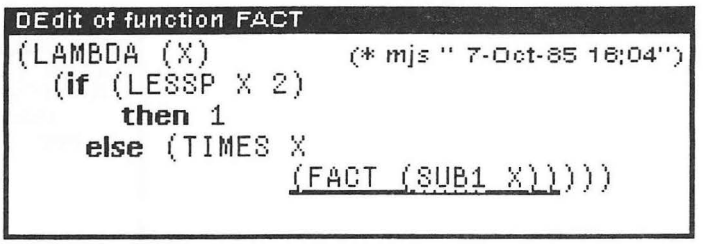
The middle button selects the containing list.

The right button extends the current selection to the lowest common ancestor of that selection and the current position.

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

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

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



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

Typing Characters to DEdit

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

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

Copy-Selection

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

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

Entering DEdit Commands

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

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

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

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

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

DEdit Functions

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

(RESETDEDIT) [Function]

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

DEdit Commands

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

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

AFTER [DEdit Command]

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

BEFORE [DEdit Command]

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

DELETE [DEdit Command]

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

REPLACE [DEdit Command]

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

SWITCH [DEdit Command]

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

() [DEdit Command]

(IN [DEdit Command]

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

) IN [DEdit Command]

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

() OUT [DEdit Command]

Removes parentheses from *TOP*.

(OUT [DEdit Command]

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

) OUT [DEdit Command]

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

UNDO [DEdit Command]

Undoes last command.

!UNDO [DEdit Command]

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

?UNDO [DEdit Command]

&UNDO [DEdit Command]

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

FIND [DEdit Command]

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

SWAP [DEdit Command]

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

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

CENTER [DEdit Command]

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

CLEAR [DEdit Command]

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

COPY [DEdit Command]

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

POP [DEdit Command]

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

REPRINT [DEdit Command]

Reprints *TOP*.

EDIT [DEdit Command]

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

Note: DEdit caches each subordinate edit window in the window from which it was entered for as long as the higher window is active. Thus, multiple DEdit commands

do not incur the cost of repeatedly allocating a new window.

EDITCOM [DEdit Command]

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

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

BREAK [DEdit Command]

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

EVAL [DEdit Command]

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

EXIT [DEdit Command]

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

OK [DEdit Command]

STOP [DEdit Command]

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

DEdit Parameters

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

EDITEMBEDTOKEN [Variable]

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

DEDITLINGER [Variable]

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

DEDITTYPEINCOMS [Variable]

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

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

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

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

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

DT.EDITMACROS [Variable]

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

(*DATATYPE MAKESOURCEFN INSTALLEDITFN*).

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

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

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

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

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

User Interface — Advanced Features

Multiple DEdit Commands

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

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

DEdit Idioms

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

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

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

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

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

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

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

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

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

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

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

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

Limitations

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