Palo Alto Research Center

A Display Oriented Programmer's Assistant

By Warren Teitelman

XEROX

A Display Oriented Programmer's Assistant

by Warren Teitelman

CSL-77-3 March 1977; Reprinted January 1981

Abstract: This paper continues and extends previous work by the author in developing systems which provide the user with various forms of explicit and implicit assistance, and in general cooperate with the user in the development of his programs. The system described in this paper makes extensive use of a bit map display and pointing device (a mouse) to significantly enrich the user's interactions with the system, and to provide capabilities not possible with terminals that essentially emulate hard copy devices. For example, any text that is displayed on the screen can be pointed at and treated as input, exactly as though it were typed, i.e., the user can say use *this* expression or *that* value, and then simply point. The user views his programming environment through a collection of display windows, each of which corresponds to a different task or context. The user can manipulate the windows, or the contents of a particular window, by a combination of keyboard inputs or pointing operations. The technique of using different windows for different tasks makes it easy for the user to manage several simultaneous tasks and contexts, e.g., defining programs, testing programs, editing, asking the system for assistance, sending and receiving messages, etc. and to switch back and forth between these tasks at his convenience.

XEROX
PALO ALTO RESEARCH CENTER
3333 Coyote Hill Road / Palo Alto / California 94304

Introduction

Lisp systems have been used for highly interactive programming for more than a decade.† During that period, much effort has been devoted to developing tools and techniques for providing powerful interactive support to the programmer. The Interlisp programming system [Tei4] represents one of the more successful projects aimed at developing a system which could be used by researchers in computer science for performing their day to day work, and could also serve as a testbed for introducing and evaluating new ideas and techniques for providing sophisticated forms of programmer assistance. Interlisp on the PDP-10 is currently used by programmers at over a dozen ARPA network sites for doing research and development on advanced artificial intelligence projects such as speech and language understanding, medical diagnosis, computer-aided instruction, automatic programming, etc. Implementations of Interlisp on several other machines are currently planned or in progress.

This paper describes a system written in Interlisp which extends the Interlisp user facilities to take advantage of a display.†† The paper is not an "idea" paper in the sense that Artificial Intelligence papers usually are. Instead, this paper describes a working system which implements and *integrates* a number of ideas and techniques previously reported in the literature by several different individuals, including the author. The idea of a display composed of multiple, overlapping regions called "windows" is attributable to and an essential part of the Smalltalk programming system designed and implemented by the Learning Research Group at Xerox Research Center [LRG]. In particular, much of the way that windows are used in the system described here was influenced by the work of Dan Ingalls on the Smalltalk user interface. The idea of using the display as a means for allowing the user to retain comprehension of complex program environments, and to monitor several simultaneous tasks, can be found in the work of Dan Swinehart [Swi]. The use of the "mouse" as a pointing device for selecting portions of a display goes back to the early work on NLS [Eng]. Finally, the techniques used for automatic error correction and the idea of having the user interact with the system through an active intermediary which maintains a history of his session, both of which appear in this paper, are parts of the standard Interlisp system [Tei1][Tei2]. The work reported in this paper is of interest primarily in how the realization of these various ideas in a single, integrated, working system dramatically confirms their value.†††

†An excellent survey of the state of the art may be found in [San].

††The author would like to acknowledge and thank R. F. Sproull and J Strother Moore, who designed and implemented critical support facilities without which this system would not have been possible, and whose ideas and intuitions provided extremely valuable guidance and inspiration during the development of the system. The form and capabilities of some of the display primitives in the current system were suggested by an earlier version of a display text facility for Interlisp designed by Terry Winograd. Finally, all of the work described herein depends heavily on the leverage provided by the Interlisp system itself, which is the result of the efforts of many individuals over a period of almost a decade, made possible by continuing ARPA support over that period.

†††When I first began work in 1969 on what was to become DWIM, the automatic error correction facility of Interlisp, by implementing a primitive spelling corrector which would automatically correct a certain class of user spelling errors, I discussed this project at length with a colleague over a period of months. One day soon after this facility was finally completed and installed in our Lisp system, this same colleague rushed to my office and in great excitement exclaimed that the system had corrected an error. I was surprised at his enthusiasm, since we had been discussing this system for months. He replied, "Yes, but it really did it!" The system described herein implements ideas that many of us have long been saying would be a good thing to have. And they really are!

Overview of the System

The system described in this paper is implemented on a version of Interlisp [Tei4] running on MAXC, a computer at the Xerox Research Center in Palo Alto. This computer emulates a PDP-10, and runs the Tenex operating system, so that from the standpoint of the user, the system he is using is Interlisp-10. The raster-scan display used by the system described in this paper is maintained by a separate 65K 16 bit word mini-computer. The minicomputer is linked to MAXC through an internal network, and implements a graphics protocol similar to the Network Graphics Protocol [Spr], but specialized for text and raster-scan images. All of the work described in this paper deals with the "high end" of the system, i.e., the user interface, and is written entirely in Interlisp.

The user communicates with the system using a standard typewriter-like keyboard. In addition, he has available a pointing device commonly called a "mouse" [Eng] used for pointing at particular locations on the screen. For those unfamiliar with this device, the mouse is a small object (about 3" by 2" by 1") with three buttons on its top. The system gives the user continuous feedback as to where it thinks the mouse is pointing by displaying a cursor on the screen. The user slides the mouse around on his working surface (causing bearings or wheels on the bottom of the mouse to rotate), and the system moves the cursor on the display. The user indicates that the mouse has arrived at some desired location by pressing one of the three buttons on the top of the mouse. The interpretation of the buttons depends on the particular program listening to the mouse. For example, when the mouse is positioned over a piece of text, and one of its buttons pressed, the corresponding text is "selected." Such selections are indicated by inverting the text, i.e., displaying it as white characters on a black background.

The user interacts with the system either by typing on the keyboard, or by pointing at commands or expressions on the screen, or an asynchronous mixture of the two. In particular, any material that is displayed on the screen can be selected and then treated as thought it were input, i.e., typed.

The ability to be able to select, i.e., point at, material currently displayed and cause it to be treated as input is extremely useful, and situations where such a facility can be used occur very often during the course of an interactive session.

Why is such a facility useful? Because most interactions with a programming system are not independent, i.e., each "event" bears some relationship to what transpired before, usually to a fairly recent event. Being able to point at (portions of) these events effectively gives the user the power of *pronoun reference*, i.e., the user can say use *this* expression or *that* value, and then simply point. This drastically reduces the amount of typing the user has to do in many situations, and results in a considerable increase in the effective "bandwidth" of the user's communication with his programming environment.

The user views his environment through a display consisting of several rectangular display "windows". Windows can be, and frequently are, overlapped on the screen. In this case, windows that are "underneath" can be brought up on top and vice versa. The resulting configuration considerably increases the user's effective working space, and also contributes to the illusion that the user is viewing a desk top containing a number of sheets of paper which the user can manipulate in various ways.

One facility provided by these windows that is not available with sheets of paper is the ability to *scroll* the window forward or backward to view material previously, but not currently, visible in the window. Thus a single window can be used to view and manipulate a body of text that would require many sheets of paper.

Each window corresponds to a different task or aspect of the user's environment. For example, there is a TYPESCRIPT window, which contains the transcript of the user's interactions with the Lisp interpreter through the programmer's assistant, a WORK AREA window which is used for editing and prettyprinting, a HISTORY window, a BACKTRACE window, a MESSAGE window, etc. Using different windows for different tasks

...makes it easy for the user to manage several simultaneous tasks and contexts, switching back and forth between them at his convenience.

Being able to switch back and forth between tasks results in a relaxed and easy style of operating more similar to the way people tend to work in the absence of restrictions. To use a programming metaphor, people operate somewhat like a collection of coroutines corresponding to tasks in various states of completion. These coroutines are continually being activated by internally and externally generated interrupts, and then suspended when higher priority interrupts arrive, e.g., a phone call that interrupts a meeting, a quick question by a colleague that interrupts a phone call, etc. Our previous experience with Interlisp supports the contention that it is of great value to the user to be able to switch back and forth quickly between related tasks. The system described in this paper makes this especially convenient, as is illustrated in the sample session presented in the body of the paper.

One technique heavily employed throughout the system is the use of *menus*. A menu is a type of window that causes a specified operation to be performed when a selection made in that window. Menus serve a number of important functions. They make it easy for the user to specify an operation without having to type. They act as a prompt for the user by providing him with a repertoire of commands from which to choose. For example, often a user will not remember the name of a command, or may not even be aware of the existence of a command.

However, most importantly, *menus greatly facilitate context switching*. As with most systems, the interpretation of the user's keystrokes (with the exception of interrupt characters which usually have a globally defined effect) depends on the state of the system. For example, when addressing the Lisp interpreter, the characters that the user types are used to construct Lisp expressions which are then evaluated. When using the editor, the characters are inserted in the indicated expression, etc. The important point is that once the user starts typing, he normally has to complete the operation or abort it. However, by selecting a menu command using the mouse, even in the midst of typing, the user can temporarily suspend the operation he is performing, go off and do something else, and then return and continue with his current context. This is also illustrated in the sample session below.

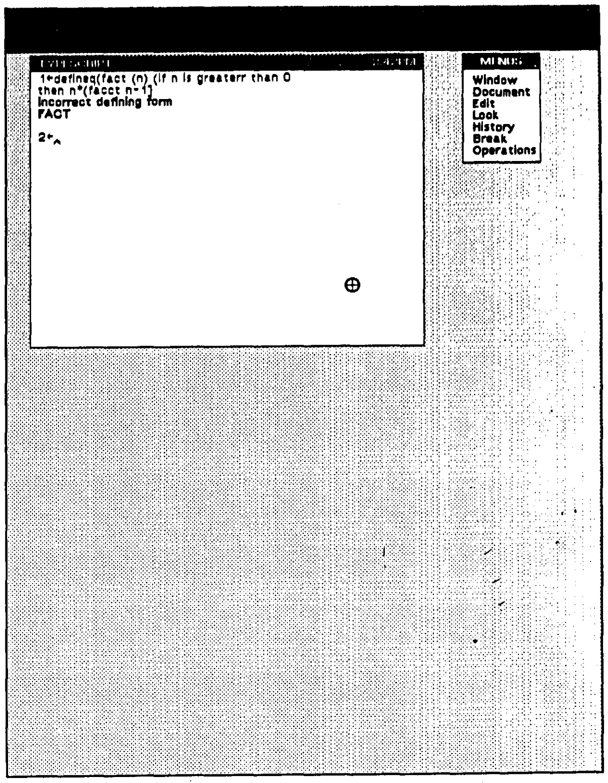
A Sample Session with the System

Since so much of the utility of the system described in this paper rest on visual effects, it is difficult to transmit the feel and smoothness of the system through words. Therefore, the form chosen for presenting the system in this paper is to take the reader through a sample session with the system, using frequent "snapshots" of the display as a substitute for the actual display itself. This session is divided into two parts. The first part is a "toy" session, in that the user is not performing any serious work. It is included only to introduce the salient features of the system. The second part of the session shows some more sophisticated use of these features in the context of an actual working session involving finding and fixing bugs, testing programs, sending and receiving messages, etc.

For readers not familiar with Lisp, please ignore Lisp related details (which we have tried to minimize). The important point is the way the system allows the user to switch back and forth between several tasks and contexts. Such a facility would be useful in any programming environment.

Sample Session—Part 1

1. Figure 1 shows the initial configuration of the screen. Three windows are displayed: the TYPESCRIPT window, which records the user's interactions with the programmer's assistant and the Lisp interpreter; the PROMPT window, which is the black region without a caption at the top of the screen used for prompting the user; and a *menu*, which is the smaller window with caption MENUS to the right of the TYPESCRIPT window. A menu is just like any other window, except that whenever a selection is made in a menu, a specified operation is also performed. This particular menu is a menu of *menus*, hence its caption. If the user selects one of its commands, each of which is the name of a menu, the corresponding menu will be displayed at the location he indicates. He can then select, and thereby perform, commands on that menu. The crosshairs shape in the lower right hand portion of the TYPESCRIPT window is the *cursor*, and indicates the current position of the mouse.



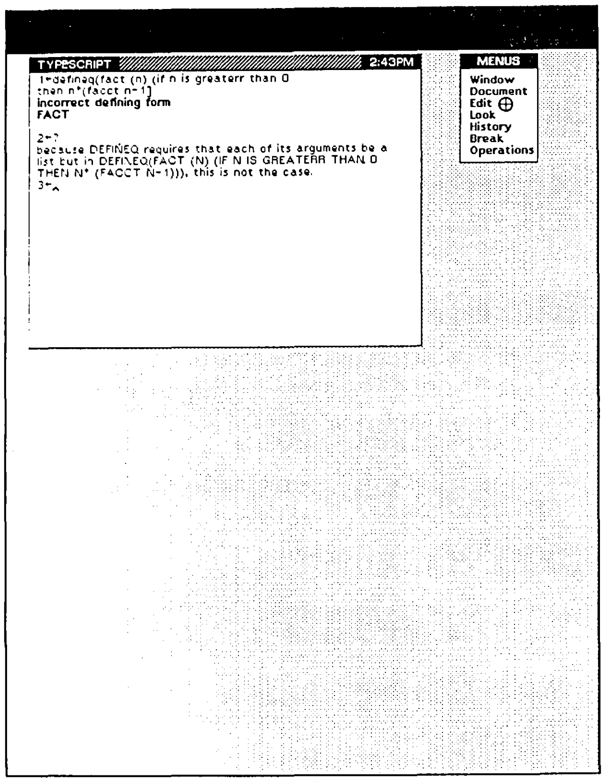
```
1←defineq(fact (n) (if n is greaterr than 0
then n*(facct n-1]
Incorrect defining form
FACT

2←
```

Figure 1

In Figure 1, I have just typed in a Lisp definition for the function FACT (factorial). Lisp has given me the error message "incorrect defining form" (displayed in bold face to set it off). The system displays a blinking caret† to indicate where the next character that I type, or the system prints, will be displayed. In Figure 1, the caret now appears immediately following the "2←", where 2 is the event number for my next interaction with the programmer's assistant, and ← is the "ready" character.

†In these figures, the caret is always shown in its "on" position.

2. I don't understand what caused this error, so I type ? to the p.a. (programmer's assistant), requesting it to supply additional explanatory information. The p.a. looks at the previous event to determine the nature of the error. In this case, using built-in information about the arguments to DEFINEQ, the p.a. tells me that the problem is that DEFINEQ encountered an atom where it expected a list, i.e., a left parentheses is missing from in front of the word "fact".† Since the programmer's assistant is maintaining a history of my interactions with the system, I don't have to retype the DEFINEQ expression. Instead, I can edit what I have already typed, and simply insert the missing left parenthesis. The EDIT menu will allow me to perform various editing operations using the mouse for pointing and the keyboard, where necessary, for supplying text. In Figure 2, I have already moved the mouse so that the cursor is positioned over the EDIT command on the MENUS menu, in preparation for "bringing up" the EDIT menu.

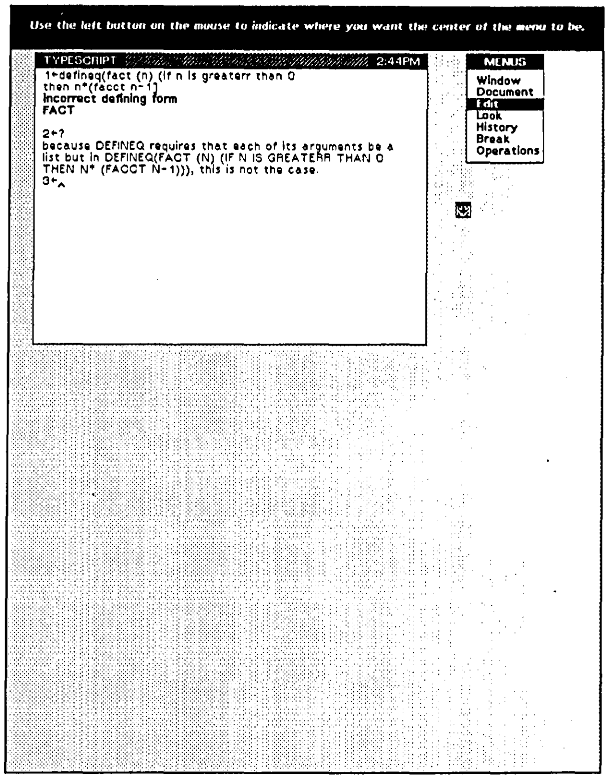


```
1←defineq(fact (n) (if n is greaterr than 0
then n*(facct n-1]
incorrect defining form
FACT

2←?
because DEFINEQ requires that each of its arguments be a
list but in DEFINEQ(FACT (N) (IF N IS GREATERR THAN 0
THEN N* (FACCT N-1))), this is not the case.
3←
```

Figure 2

†If the p.a. did not know anything about this particular error, it would refer to the index of the on-line Interlisp Reference Manual and present the corresponding text associated with the error message by way of explanation. The user can also augment the built-in information that the p.a. has about system functions by informing the p.a. about the requirements of his own functions. He can then use the ? command to explain errors in his own programs.

3. I press a button on the mouse to select the EDIT command in the MENUS menu. The system indicates the selection by displaying EDIT as white on black. The PROMPT window tells me to use the left button on the mouse to indicate where I want the center of the (EDIT) menu to appear. The cursor is changed to an icon of a menu with a cross in its center to suggest the operation that is pending. At this point, I don't *have* to complete this operation. I can type in other expressions to the programmer's assistant, perform other menu operations, etc. The process which is waiting for me to supply the indicated information is simply a co-routine which has been suspended.† However, since I want to fix up the DEFINEQ expression before going on to anything else, I move the cursor to the position at which I want the EDIT menu to appear, which is below the MENUS menu and to the right of the TYPESCRIPT window, as shown in Figure 3.

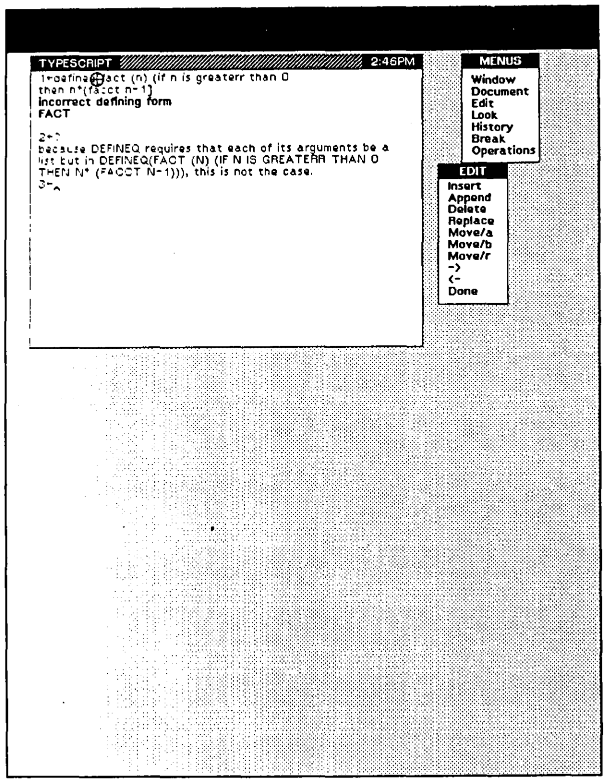


Use the left button on the mouse to indicate where you want the center of the menu to be.

```
1←defineq(fact (n) (if n is greaterr than 0
then n*(facct n-1]
incorrect defining form
FACT

2←?
because DEFINEQ requires that each of its arguments be a
list but in DEFINEQ(FACT (N) (IF N IS GREATERR THAN 0
THEN N* (FACCT N-1))), this is not the case.
3←
```

Figure 3

4. I press the left button on the mouse, causing the EDIT menu to appear at the location of the cursor. In this position, the EDIT menu slightly overlaps both the TYPESCRIPT window and the MENUS menu, so the system automatically adjusts the EDIT menu by sliding it off these windows to its location as shown in Figure 4.††



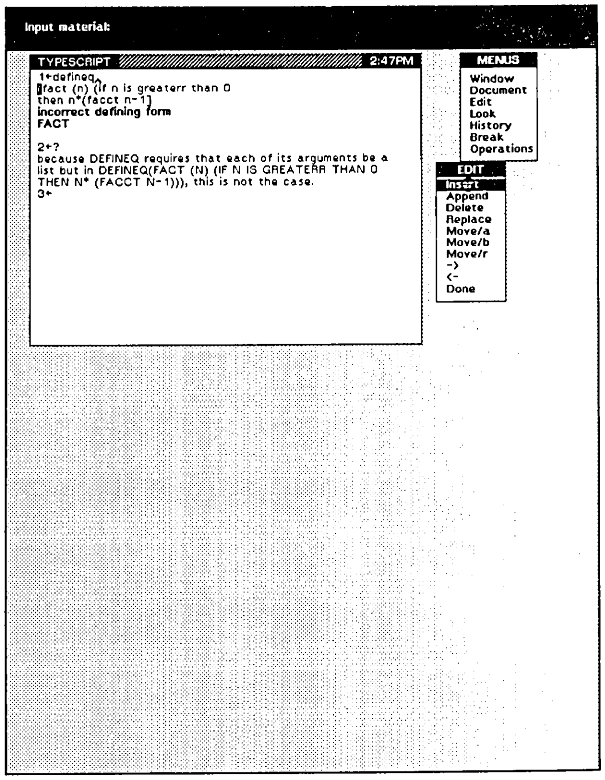
```
1←defineq(fact (n) (if n is greaterr than 0
then n*(facct n-1]
incorrect defining form
FACT

2←?
because DEFINEQ requires that each of its arguments be a
list but in DEFINEQ(FACT (N) (IF N IS GREATERR THAN 0
THEN N* (FACCT N-1))), this is not the case.
3←
```

Figure 4

†See description of the "Spaghetti Stack" facility in [Bob] and [Tei4].

††I could force the EDIT menu to overlap the TYPESCRIPT window by positioning it exactly using one of the commands on the WINDOW menu. However, since in this case I only positioned the menu approximately, the system tries to "Do What I Mean", a philosophy of system design we have tried to follow throughout the Interlisp system [Teil]..

5. Now I am ready to edit. I select the left parenthesis in the first line of the TYPESCRIPT window, and then select the INSERT command on the EDIT menu. The line of text in the TYPESCRIPT window is broken just before the selection (the left parenthesis), and the caret is moved to that location. The PROMPT window instructs me to input material. Anything I type will appear at the location indicated by the caret.

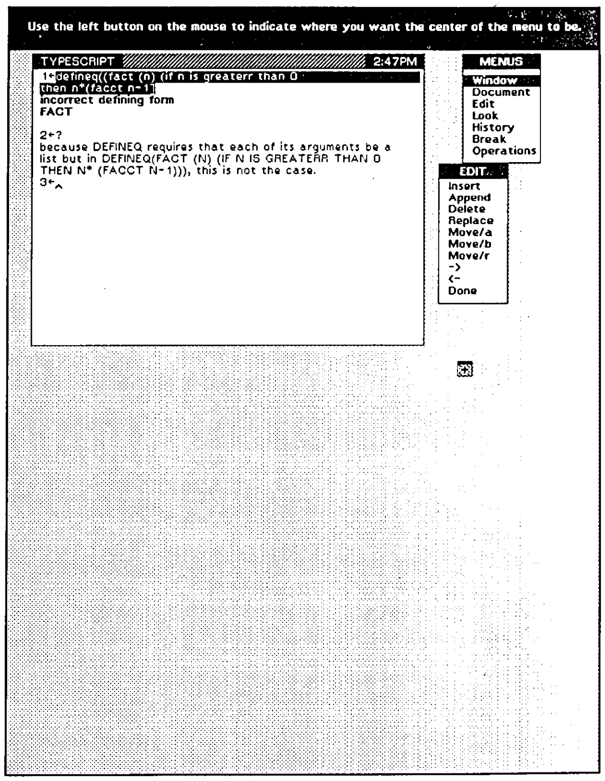


Input material:

```
TYPESCRIPT                                   2:47PM
1←defineq
(fact (n) (if n is greaterr than 0
then n*(facct n-1]
incorrect defining form
FACT

2←?
because DEFINEQ requires that each of its arguments be a
list but in DEFINEQ(FACT (N) (IF N IS GREATERR THAN 0
THEN N* (FACCT N-1))), this is not the case.
3←
```

Figure 5

6. I type in a single left parenthesis, and terminate the INSERT operation. The line of text I have been editing is rejoined, and the caret returned to the appropriate location at the end of the TYPESCRIPT window. I now want to cause the corrected text to be *re-input* in order to perform my original operation, i.e., define my function. Therefore, I select the text by first selecting the "d" in "defineq" and then extending this selection through the final "]". Then, using the same method as previously shown for bringing up the EDIT menu, I bring up the WINDOW menu in order to obtain the command for inputting selected material.

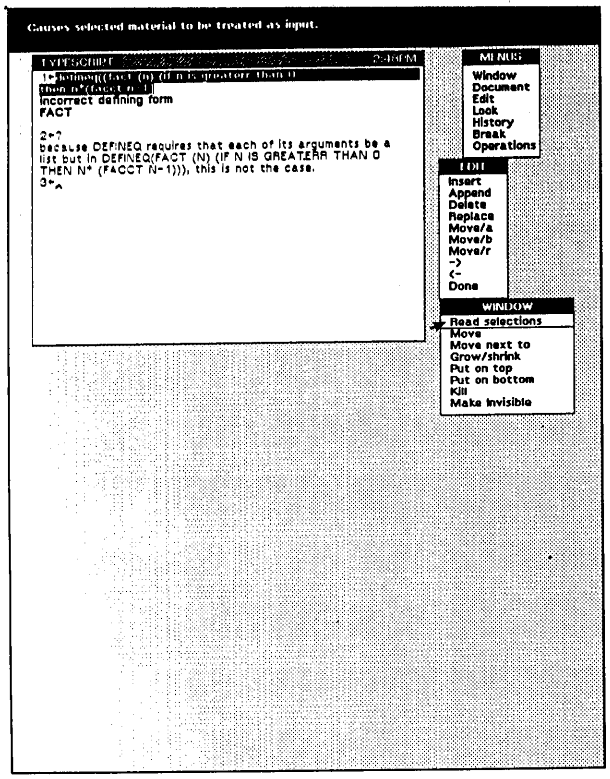


Use the left button on the mouse to indicate where you want the center of the menu to be.

```
TYPESCRIPT                                   2:47PM
1←defineq((fact (n) (if n is greaterr than 0
then n*(facct n-1]
incorrect defining form
FACT

2←?
because DEFINEQ requires that each of its arguments be a
list but in DEFINEQ(FACT (N) (IF N IS GREATERR THAN 0
THEN N* (FACCT N-1))), this is not the case.
3←
```

Figure 6

7. The WINDOW menu contains the command "READ SELECTIONS" which is the command that I *think* does what I want. I therefore select this command, but instead of *clicking* the mouse button, I *hold* the mouse button down. This instructs the system to tell me what it *would* do if this operation were actually performed. Here, the PROMPT window informs me that the "READ SELECTIONS" command causes the selected material to be treated as input. Figure 7 shows the display as of this point. The cursor has been changed to an arrow to indicate that a selection is about to be made. The material that would be selected, namely the "READ SELECTIONS" command, is underscored. If I want to perform this selection, I simply release the mouse button. Otherwise, I can move the mouse to another location and release it there in order to perform a selection at the new location, or move it off of the menu entirely to abort the selection.


Causes selected material to be treated as input.

```
TYPESCRIPT                                   2:48PM
1←defineq((fact (n) (if n is greaterr than 0
then n*(facct n-1]
incorrect defining form
FACT

2←?
because DEFINEQ requires that each of its arguments be a
list but in DEFINEQ(FACT (N) (IF N IS GREATERR THAN 0
THEN N* (FACCT N-1))), this is not the case.
3←^
```

Figure 7

8. I release the mouse button, and the selected material is treated exactly as though I had typed it, i.e., becomes event number 3 and causes the function FACT to be defined. As mentioned before, this ability of being able to select, i.e., point at, material currently displayed and cause it to be treated as input is extremely useful, and the situations where such a facility can be used occur very often during the course of an interactive session.

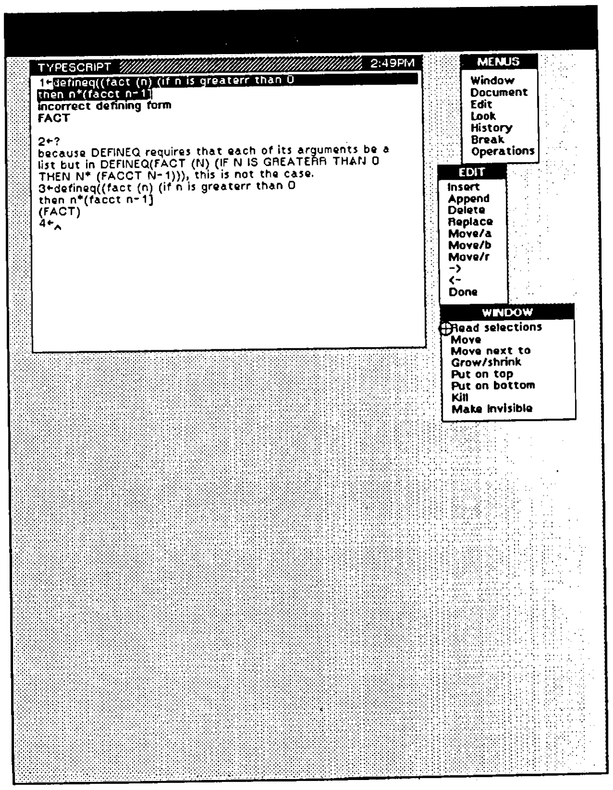
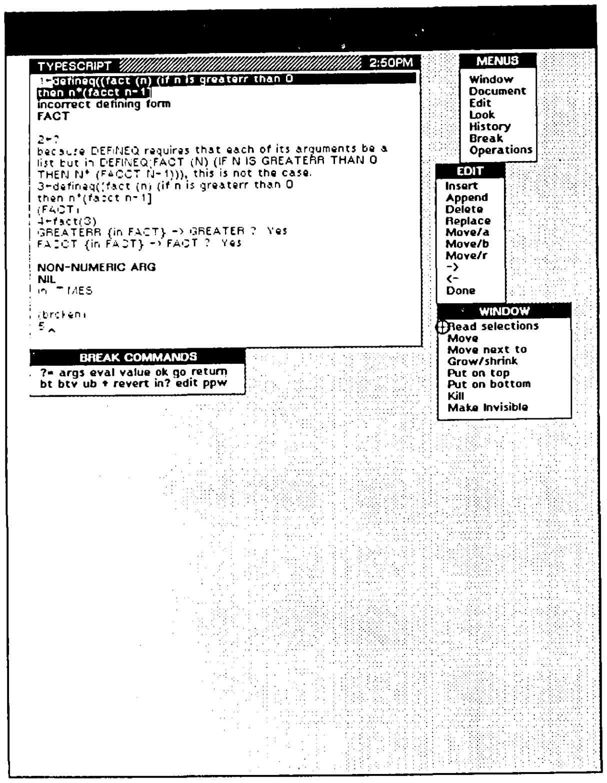

```
TYPESCRIPT                                   2:49PM
1←defineq((fact (n) (if n is greaterr than 0
then n*(facct n-1]
incorrect defining form
FACT

2←?
because DEFINEQ requires that each of its arguments be a
list but in DEFINEQ(FACT (N) (IF N IS GREATERR THAN 0
THEN N* (FACCT N-1))), this is not the case.
3←defineq((fact (n) (if n is greaterr than 0
then n*(facct n-1]
(FACT)
4←^
```

Figure 8

9. I now try out my function by typing FACT(3). At this point, CLISP [Tei3] is invoked to translate the if-then expression in the definition of FACT into an equivalent Lisp construct. CLISP runs into a problem regarding the word GREATERR, and DWIM offers a spelling correction. I type Y (the spelling corrector supplies the "es"), and the correction is made. I had also misspelled the recursive call to FACT in the body of the definition of FACT. Since the programmer's assistant "noticed" this new function, i.e., FACT, when I first defined it, DWIM is able to suggest the correction of FACCT to FACT, which I also confirm. Figure 9 shows the display after these two corrections have been made. At this point, the definition of FACT has been translated to Lisp successfully, at least from a syntactic standpoint, and an error is encountered which DWIM cannot handle. The error message NON-NUMERIC ARG NIL is printed, and Interlisp goes into a break. A menu of break commands automatically appears just below the TYPESCRIPT window.

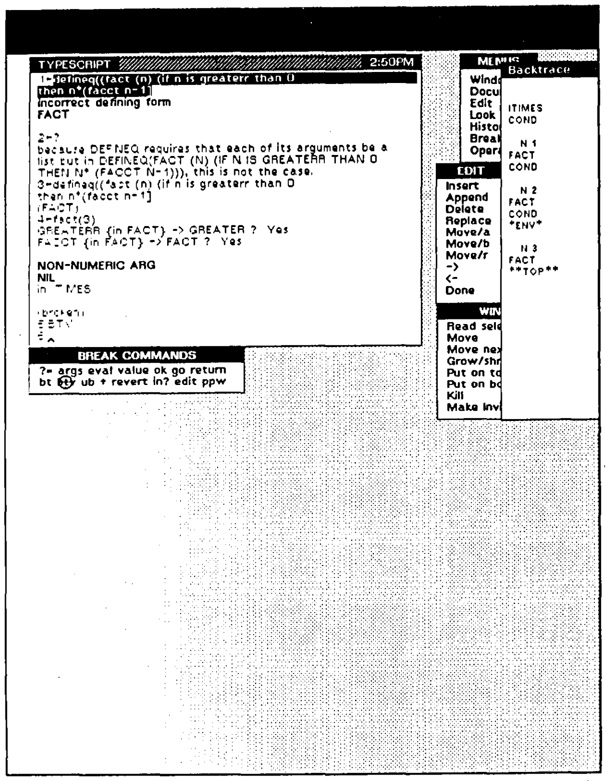


```
1←defineq((fact (n) (if n is greaterr than 0
then n*(facct n-1]
incorrect defining form
FACT

2←?
because DEFINEQ requires that each of its arguments be a
list but in DEFINEQ(FACT (N) (IF N IS GREATERR THAN 0
THEN N* (FACCT N-1))), this is not the case.
3←defineq((fact (n) (if n is greaterr than 0
then n*(facct n-1]
(FACT)
4←fact(3)
GREATERR {in FACT} -> GREATER ?  Yes
FACCT {in FACT} -> FACT ?  Yes

NON-NUMERIC ARG
NIL
in ITIMES

(broken)
5←
```

Figure 9

At this point the user is once again addressing the Lisp interpreter through the programmer's assistant. However, the context of his computation has been preserved and is available so that the user can, for example, examine the values of locally bound variables, see the control structure that lead to this point in the computation, etc., and if he wishes, fix or bypass the problem and continue the computation. This capability is most important for interactive debugging [Teil]. In this particular case, the arithmetic operation MULTIPLY (as implemented by the Lisp function ITIMES) is waiting for a number, i.e., the value of the break will be used as a multiplicand. In effect, *the system has called the user as a subroutine to supply this number.*

10. I select the BTV command, requesting a backtrace of function names along with the names and values of the bound variables for each corresponding function call. The backtrace is printed in a separate BACKTRACE window, which is automatically displayed when the backtrace command is invoked. The BACKTRACE window is shown at the right of the screen in Figure 10. Note that it overlaps the three menus. However, I can still perform operations using those menus by pointing at the part of the menu that is visible. I can select elements in the BACKTRACE window to focus the attention of the break package on a particular frame, e.g., to evaluate an expression in a different context, to cause the computation to revert back to that point, etc. The backtrace shows me that I am under my function FACT, and that it made three recursive calls before the error, with N being decremented by 1 each call, so it looks like FACT is recursing properly.

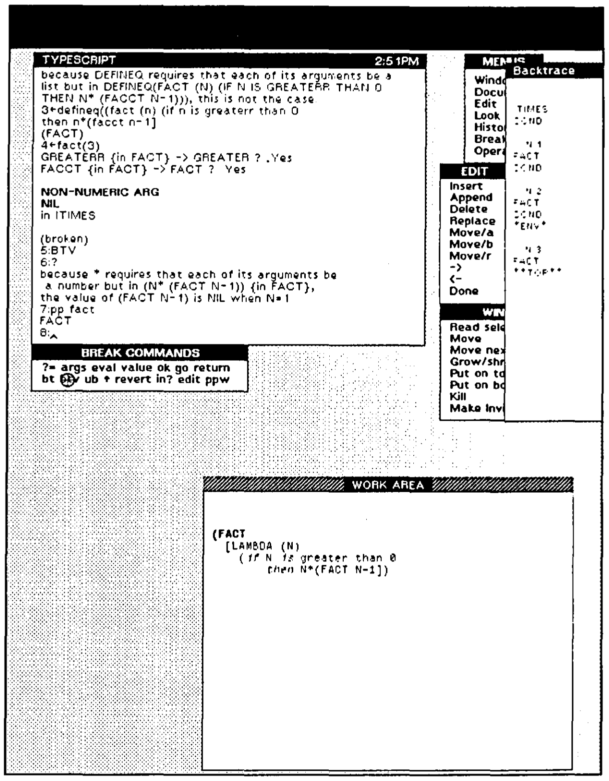


```
1←defineq((fact (n) (if n is greaterr than 0
then n*(facct n-1]
incorrect defining form
FACT

2←?
because DEFINEQ requires that each of its arguments be a
list but in DEFINEQ(FACT (N) (IF N IS GREATERR THAN 0
THEN N* (FACCT N-1))), this is not the case.
3←defineq((fact (n) (if n is greaterr than 0
then n*(facct n-1]
(FACT)
4←fact(3)
GREATERR {in FACT} -> GREATER ?  Yes
FACCT {in FACT} -> FACT ?  Yes

NON-NUMERIC ARG
NIL
in ITIMES

(broken)
5:BTV
6:^
```

```
ITIMES
COND

  N 1
FACT
COND

  N 2
FACT
COND
*ENV*

  N 3
FACT
**TOP**
```

Figure 10

11. I still don't understand why the error occurred, so I try typing the ? command again. In this case, the programmer's assistant tells me that the problem is that one of the operands to * (the MULTIPLY operator) was (FACT N-1) and that the value of (FACT N-1) is NIL when N=1. In other words, when FACT is called with N=0, it returns NIL. The p.a. is able to generate this explanation because (1) it knows that all of the arguments to * must be numbers, and (2) it can examine the state of the computation on the stack. In this case, it found that the second operand to ITIMES was NIL, which is not a number, and that the expression that produced this particular value was (FACT N-1) in the expression (N*(FACT N-1)) which is contained in the function FACT, and that at the time this call occurred, the value of N was 1.



```
because DEFINEQ requires that each of its arguments be a
list but in DEFINEQ(FACT (N) (IF N IS GREATERR THAN 0
THEN N* (FACCT N-1))), this is not the case.
3←defineq((fact (n) (if n is greaterr than 0
then n*(facct n-1]
(FACT)
4←fact(3)
GREATERR {in FACT} -> GREATER ? .Yes
FACCT {in FACT} -> FACT ?  Yes

NON-NUMERIC ARG
NIL
in ITIMES

(broken)
5:BTV
6:?
because * requires that each of its arguments be
 a number but in (N* (FACT N-1)) {in FACT},
the value of (FACT N-1) is NIL when N=1
7:pp fact
FACT
8:^
```

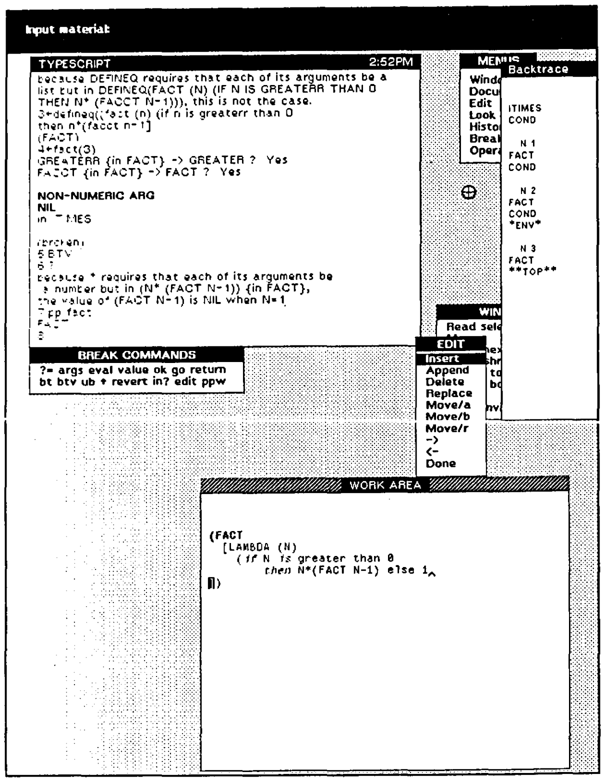
```
ITIMES
COND

  N 1
FACT
COND

  N 2
FACT
COND
*ENV*

  N 3
FACT
**TOP**
```

```
(FACT
  [LAMBDA (N)
    (if N is greater than 0
        then N*(FACT N-1])
```

Figure 11

I now realize that the problem is simply that I neglected to specify the value of FACT for N=0.† Therefore, I prettyprint the definition of FACT in preparation for editing it. Figure 11 shows the definition of FACT prettyprinted in my WORK AREA window, which automatically appeared when prettyprint was called. Note that the definition of FACT now shows the two misspelled words, GREATERR and FACCT, spelled correctly.

12. I select the right square bracket in the definition of FACT in the WORK AREA window, and then select the INSERT comand on the EDIT menu. The EDIT menu automatically moves so as to be close to the window that I am editing. I make the necessary correction by typing ") ELSE 1", i.e., if N is not greater than 0, FACT should return 1. Figure 12 shows the display just before I complete the INSERT. Note that the caret appears in the WORK AREA window where I am typing. The cursor is in the upper right hand portion of the screen at the location of the INSERT command before the EDIT menu moved to be close to the WORK AREA.



```
because DEFINEQ requires that each of its arguments be a
list but in DEFINEQ(FACT (N) (IF N IS GREATERR THAN 0
THEN N* (FACCT N-1))), this is not the case.
3←defineq((fact (n) (if n is greaterr than 0
then n*(facct n-1]
(FACT)
4←fact(3)
GREATERR {in FACT} -> GREATER ?  Yes
FACCT {in FACT} -> FACT ?  Yes

NON-NUMERIC ARG
NIL
in ITIMES

(broken)
5 BTV
6 ?
because * requires that each of its arguments be
a number but in (N* (FACT N-1)) {in FACT},
the value of (FACT N-1) is NIL when N=1
7 pp fact
FACT
8
```

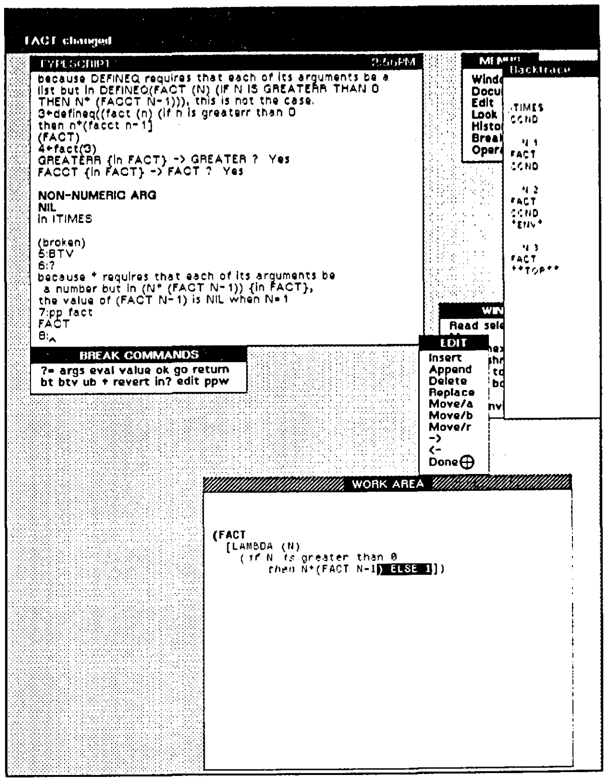
```
ITIMES
COND

  N 1
FACT
COND

  N 2
FACT
COND
*ENV*

  N 3
FACT
**TOP**
```

```
(FACT
  [LAMBDA (N)
    (if N is greater than 0
        then N*(FACT N-1) else 1^
])
```

Figure 12

†In Interlisp, if none of the predicates of an if-then expression evaluate true, the value of the expression defaults to NIL.

13. I complete the INSERT, and then select the DONE command on the EDIT menu to indicate that I am finished editing this expression. The PROMPT window reports that the definition of FACT has been changed. Note that I did not *have* to finish editing FACT at this point: I could have typed in expressions to be evaluated, performed other menu operations, etc., even edited other expressions, before selecting the DONE command for this expression. This is another example of being able to suspend different tasks in varying states of completion and go back to them at some later point.



```
FACT changed
```

```
TYPESCRIPT                                        2:56PM
because DEFINEQ requires that each of its arguments be a
list but in DEFINEQ(FACT (N) (IF N IS GREATERR THAN 0
THEN N* (FACCT N-1))), this is not the case.
3←defineq((fact (n) (if n is greaterr than 0
then n*(facct n-1]
(FACT)
4←fact(3)
GREATERR {in FACT} -> GREATER ?  Yes
FACCT {in FACT} -> FACT ?  Yes

NON-NUMERIC ARG
NIL
in ITIMES

(broken)
5:BTV
6:?
because * requires that each of its arguments be
 a number but in (N* (FACT N-1)) {in FACT},
the value of (FACT N-1) is NIL when N=1
7:pp fact
FACT
8:
```

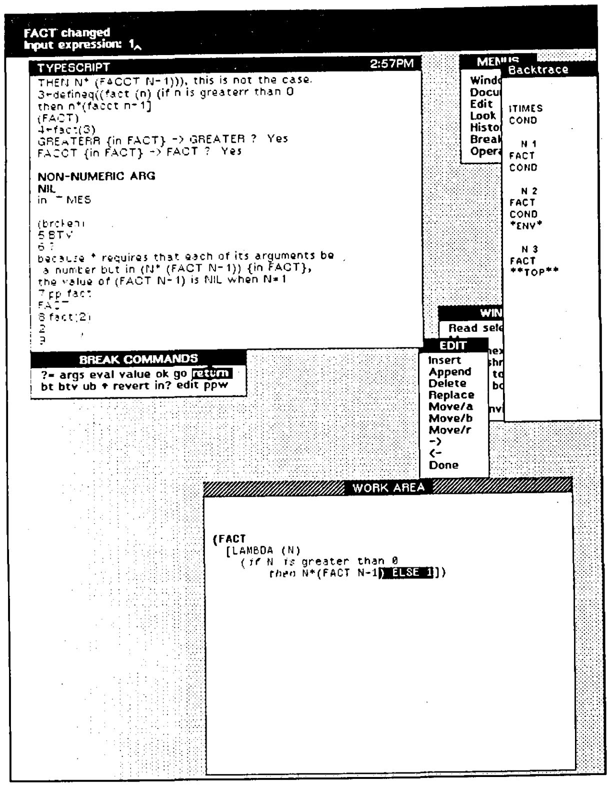
```
WORK AREA
(FACT
  [LAMBDA (N)
    (if N is greater than 0
        then N*(FACT N-1) ELSE 1])
```

Figure 13

14. I now test out my change by typing fact(2), which works correctly. Now I want to *continue* with the computation. Note that I am still in the original break that followed the error. The arithmetic operation * (i.e., the Lisp function ITIMES) is still waiting for a number to be used as a multiplicand. I therefore select the RETURN command on the BREAK menu. The PROMPT window tells me to INPUT EXPRESSION and the caret moves to the PROMPT window. I type 1 as the value to be returned from this error break. Figure 14 shows the display at this point just after I type 1, which is echoed (displayed) in the PROMPT window.

Note: in actual practice, for a computation as trivial as FACT(3), I would probably simply reset (abort back to the top) and reexecute FACT(3) rather than bothering to continue the computation, since so little has been invested in getting to this point. However,

being able to continue a computation following an error is especially useful when an error occurs following a significant amount of computation, or when the computation has left things in an "unclean state" as a result of global side effects. Such a facility is also essential for good interactive debugging.



```
FACT changed
Input expression: 1
```

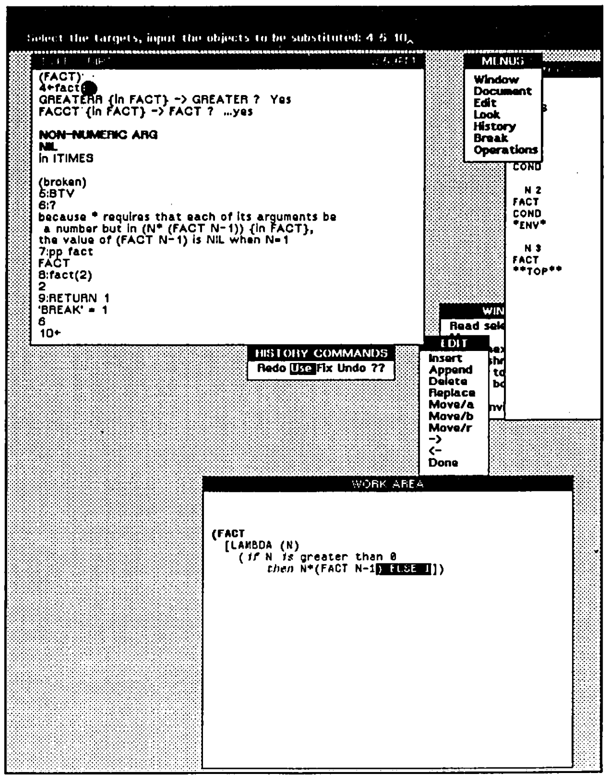
```
TYPESCRIPT                                        2:57PM
THEN N* (FACCT N-1))), this is not the case.
3←defineq((fact (n) (if n is greaterr than 0
then n*(facct n-1]
(FACT)
4←fact(3)
GREATERR {in FACT} -> GREATER ?  Yes
FACCT {in FACT} -> FACT ?  Yes

NON-NUMERIC ARG
NIL
in ITIMES

(broken)
5:BTV
6:?
because * requires that each of its arguments be
 a number but in (N* (FACT N-1)) {in FACT},
the value of (FACT N-1) is NIL when N=1
7:pp fact
FACT
8:fact(2)
2
9:
```

```
WORK AREA
(FACT
  [LAMBDA (N)
    (if N is greater than 0
        then N*(FACT N-1) ELSE 1])
```

Figure 14

15. I complete typing the expression for the RETURN command, thereby causing 1 to be returned as the value of the break, which causes (1 * 1) to be computed and returned as the value of FACT(1), which then causes (2 * 1) to be computed, etc., and finally the original computation of FACT(3) finishes and returns 6 as its value as shown in Figure 15, in the next to the bottom line of the TYPESCRIPT window. The BREAK menu has disappeared since we are no longer in a break.

I now want to try FACT on some other values, so I bring up the HISTORY menu, and select the USE command, which is a command to the programmer's assistant to reexecute a previous event, or events, with new values. The PROMPT window instructs me to select the targets and to input the objects to be substituted. I select the "3" in FACT(3) (near the top of the TYPESCRIPT window) and input "4 5 10" (echoed in the PROMPT window), i.e., I am requesting that FACT(4), FACT(5) and FACT(10) be computed.


Select the targets, input the objects to be substituted: 4 5 10

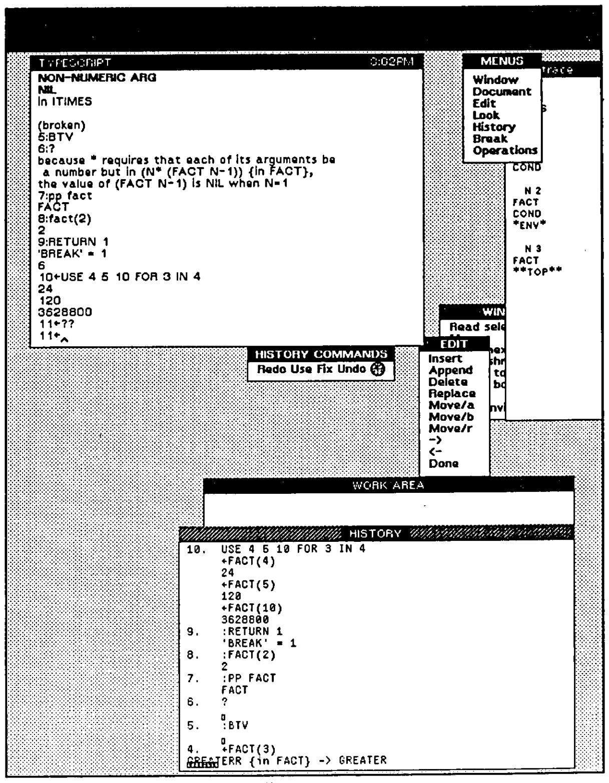
```
(FACT)
4←fact(3)
GREATERR {in FACT} -> GREATER ?  Yes
FACCT {in FACT} -> FACT ?  ...yes

NON-NUMERIC ARG
NIL
in ITIMES

(broken)
5:BTV
6:?
because * requires that each of its arguments be
 a number but in (N* (FACT N-1)) {in FACT},
the value of (FACT N-1) is NIL when N=1
7:pp fact
FACT
8:fact(2)
2
9:RETURN 1
'BREAK' = 1
6
10←
```

```
(FACT
  [LAMBDA (N)
    (if N is greater than 0
        then N*(FACT N-1) ELSE 1])
```

Figure 15

16. The resulting history operation is equivalent to typing USE 4 5 10 FOR 3 IN 4,† which the p.a. prints in the TYPESCRIPT window to show me what is happening. This USE command now causes three computations to be performed, corresponding to the result of substituting 4 for 3 in FACT(3), the result of substituting 5 for 3 in FACT(3), and the result of substituting 10 for 3 in FACT(3). The values produced by these three computations, 24, 120, and 3628800, are printed in the TYPESCRIPT window, as shown in Figure 16. Finally, I ask for a replay of the history of my session, by selecting the ?? command in the HISTORY menu. The HISTORY window is brought up, and the history of my session, in reverse chronological order, is printed in this window, as shown in Figure 16.†

†4 is the event number of the event corresponding to FACT(3).


```
TYPESCRIPT                                   3:02PM
NON-NUMERIC ARG
NIL
in ITIMES

(broken)
5:BTV
6:?
because * requires that each of its arguments be
 a number but in (N* (FACT N-1)) {in FACT},
the value of (FACT N-1) is NIL when N=1
7:pp fact
FACT
8:fact(2)
2
9:RETURN 1
'BREAK' = 1
6
10←USE 4 5 10 FOR 3 IN 4
24
120
3628800
11←??
11←
```

```
HISTORY
10.  USE 4 5 10 FOR 3 IN 4
     ←FACT(4)
     24
     ←FACT(5)
     120
     ←FACT(10)
     3628800
9.   :RETURN 1
     'BREAK' = 1
8.   :FACT(2)
     2
7.   :PP FACT
     FACT
6.   ?

5.   :BTV

4.   ←FACT(3)
GREATERR {in FACT} -> GREATER
```

Figure 16

This completes the "toy" session designed to illustrate some of the basic features of the system. Note that at this point the display contains nine different windows. Five of these windows are control windows (menus). The other four windows describe various processes. Note that the windows have not been a burden on the user: he does not "manage" the windows, although he could perform explicit operations on them such as changing their position, or size, or shape, or editing their contents as we have seen. The feeling to the user is that the windows more or less manage themselves, and this contributes greatly to the smoothness of the system.

Example Session—Part 2

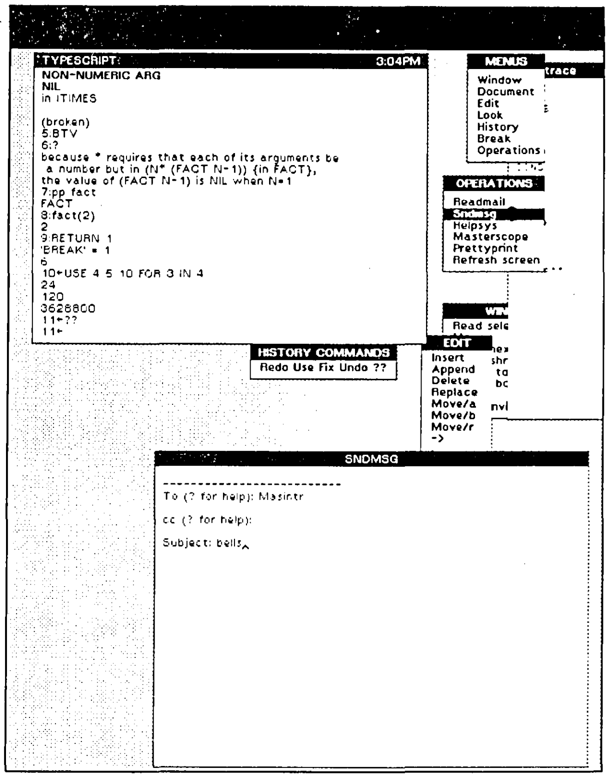
However, to really appreciate the power of the system, one must see how the various facilities, and the user, interact in a real working situation. The following session, which is a continuation of the above session, illustrates this. In addition to the prettyprinting, editing, break, and history facilities illustrated earlier, several other important Interlisp facilities are introduced during the course of the session below, such as Helpsys, which interfaces with the on-line Interlisp Reference Manual to allow the user to ask questions and see explanatory material from the manual, and Masterscope, a sophisticated interactive program for analyzing and cross referencing user programs. Masterscope then allows the user to interrogate the resulting data base both with respect to the control structure of his programs, i.e., who calls/is called by whom, and to their data structure, i.e., where variables are bound, set, or referenced, or which functions use particular record declarations. These facilities are necessarily used in a fairly simple and straightforward fashion in the session below. However, the important point to observe is how the display together with multiple windows enables the user to call up the various packages quickly and easily and then to dismiss them when he is finished, all with minimal interference with his context, i.e., the *user's* context, not that of his program.

Receiving and sending messages from other users play an important part in this session, as they do in real life applications. The system described in this paper makes it especially easy to process messages because the reading and sending of messages is implemented *within the Interlisp system*, instead of in a separate subsystem. Thus, the user does not have to give up his context in order to process his mail. Furthermore, since the message facilities are now part of the Lisp environment and vice versa, the user can obtain material from messages that he receives and *insert it directly into his own programs* or *evaluate the corresponding expressions* by using the READ SELECTIONS command described earlier. Conversely, the user can insert material from his own environment, or from messages that he has received since they are also a part of his environment, into messages that are to be sent. Both of these facilities are extremely useful.

Note: the session presented below is "canned" in that the events described did not actually occur so fortuitously in a single session, nor in such a nice sequence for the purposes of demonstrating the system. However, the session is genuine in that the figures that accompany the text are in fact actual snapshots of the display taken in sequence through a session in which the indicated operations were performed. And, in fact, the events described were culled from actual sessions over the course of several months, i.e., I really did find the bugs, make the changes, and receive and send the messages depicted below.

†In addition to seeing a replay of his history, the user can also *scroll* the (contents of the) TYPESCRIPT window backwards in time to see the transcript of earlier interactions with the system. The difference between the history and the TYPESCRIPT is that the TYPESCRIPT contains a record of all characters input or output, e.g., includes messages printed by the system and by the user's programs. The history contains a subset of these characters, organized according to *events*. For example, 6, the value returned by FACT(3), actually appears 18 lines below FACT(3) in the TYPESCRIPT window, but in the HISTORY window, it would be shown as the value of event number 4, regardless of the fact that events 5 thru 9 occurred between the time that event 4 was begun and the time it completed.

17. I observe an anomaly in my history window as shown in Figure 16: a sequence of bells (displayed as little boxes) in the middle of my history. When the history was actually printed, the system paused at this point and seemed to be waiting for me to type something. I decide to ask a colleague about this. I therefore bring up the OPERATIONS menu and select the SNDMSG command. SNDMSG brings up its own window, and asks me who the message is to. I respond "Masintr" (misspelled—his name is actually Masinter). SNDMSG then asks whether I want any "carbon copies" sent to other recipients, and I simply type a carriage return, since I don't. SNDMSG then asks what the subject of the message is, and I type "bells". Figure 17 shows the display at this point. The caret is in the SNDMSG window, immediately following the word "bells".†

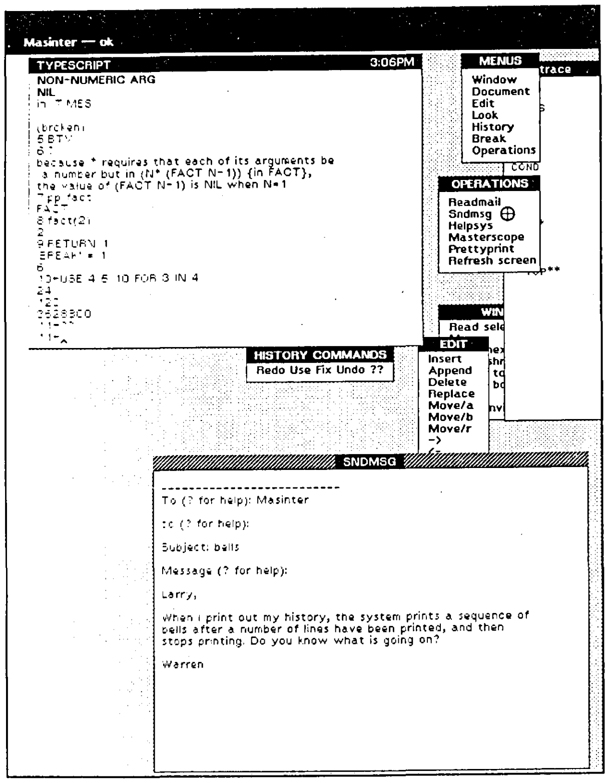


```
NON-NUMERIC ARG
NIL
in ITIMES

(broken)
5:BTV
6:?
because * requires that each of its arguments be
 a number but in (N* (FACT N-1)) {in FACT},
the value of (FACT N-1) is NIL when N=1
7:pp fact
FACT
8:fact(2)
2
9:RETURN 1
'BREAK' = 1
6
10←USE 4 5 10 FOR 3 IN 4
24
120
3628800
11←??
11←
```

```
--------------------------
To (? for help): Masintr

cc (? for help):

Subject: bells
```

Figure 17

18. I complete typing the message, and terminate with a control-Z to indicate that I want the message sent.†† SNDMSG does not recognize the recipient, "Masintr",. and calls the spelling corrector, which returns the correct spelling, "Masinter".††† The message header is fixed accordingly, and the system informs me in the PROMPT window that the message has been sent.



```
NON-NUMERIC ARG
NIL
in ITIMES

(broken)
5:BTV
6:?
because * requires that each of its arguments be
 a number but in (N* (FACT N-1)) {in FACT},
the value of (FACT N-1) is NIL when N=1
7:pp fact
FACT
8:fact(2)
2
9:RETURN 1
'BREAK' = 1
6
10←USE 4 5 10 FOR 3 IN 4
24
120
3628800
11←??
11←
```

```
--------------------------
To (? for help): Masinter

cc (? for help):

Subject: bells

Message (? for help):

Larry,

When I print out my history, the system prints a sequence of
bells after a number of lines have been printed, and then
stops printing. Do you know what is going on?

Warren
```

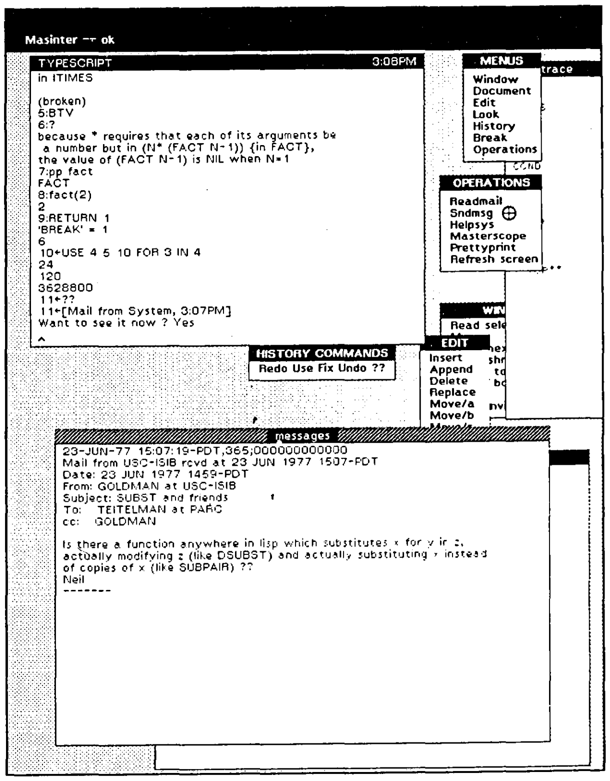
Figure 18

†Although this SNDMSG looks like the SNDMSG facility provided by Tenex, it is entirely a part of and written in the Interlisp system.

††The Interlisp version of SNDMSG adheres to the Tenex convention for sending messages.

†††Both SNDMSG and READMAIL are "watching" what I am doing, and build a list of those users that I exchange messages with. When I misspelled Masintr, the spelling corrector was called with this list, and was able to perform the correction. Had it failed, I would have been informed and allowed to intervene, as shown later in the session.

19. At this point, I am informed in the TYPESCRIPT window by the programmer's assistant that there is [Mail from System], i.e. some mail has arrived for me. I had previously instructed the p.a. to periodically check whether any new mail had arrived, and to so inform me. Since I am not doing anything at this point except waiting for Masinter to respond to my message, I elect to see my mail, type Yes to the question "Want to see it now ?", and the message is displayed in the MESSAGES window. The message is from a user of Interlisp at the Information Sciences Institute in Los Angeles with a question about Interlisp.†



```
Masinter -- ok
```

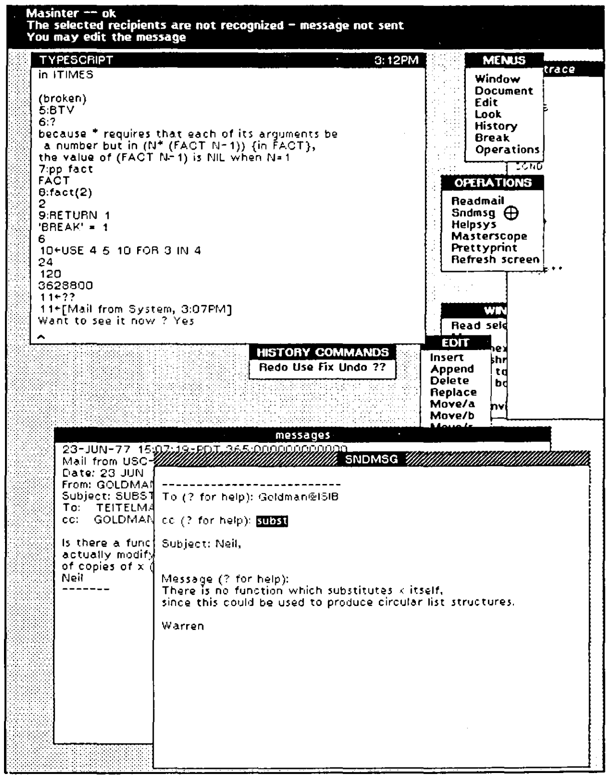
```
TYPESCRIPT                                         3:08PM
in ITIMES

(broken)
5:BTV
6:?
because * requires that each of its arguments be
 a number but in (N* (FACT N-1)) {in FACT},
the value of (FACT N-1) is NIL when N=1
7:pp fact
FACT
8:fact(2)
2
9:RETURN 1
'BREAK' = 1
6
 10←USE 4 5 10 FOR 3 IN 4
24
 120
3628800
 11←??
 11←[Mail from System, 3:07PM]
Want to see it now ? Yes
^
```

```
messages
23-JUN-77 15:07:19-PDT,365;000000000000
Mail from USC-ISIB rcvd at 23 JUN 1977 1507-PDT
Date: 23 JUN 1977 1459-PDT
From: GOLDMAN at USC-ISIB
Subject: SUBST and friends
To:   TEITELMAN at PARC
cc:   GOLDMAN

Is there a function anywhere in lisp which substitutes x for y in z,
actually modifying z (like DSUBST) and actually substituting x instead
of copies of x (like SUBPAIR) ??
Neil
-------
```

Figure 19

20. I decide to respond to the message from Goldman right now, and select the SNDMSG command in the OPERATIONS menu. The SNDMSG window comes back on top, and I type my response. I finish the message without noticing that I had inadvertently typed the subject of the message in the cc (carbon copies) field, and the begining of the message in the subject field. SNDMSG was unable to interpret the word "Subst" as a message recipient, and informs me in the PROMPT window that the message wasn't sent.

†Both Xerox PARC and the Information Sciences Institute are hosts on the ARPA network. The mail facilities supported by the various hosts of the network enable users at any site to exchange messages with users at any other site. Such "electronic" mail has become the preferred form of communication for questions, bug reports, suggestions, etc.



```
Masinter -- ok
The selected recipients are not recognized - message not sent
You may edit the message
```

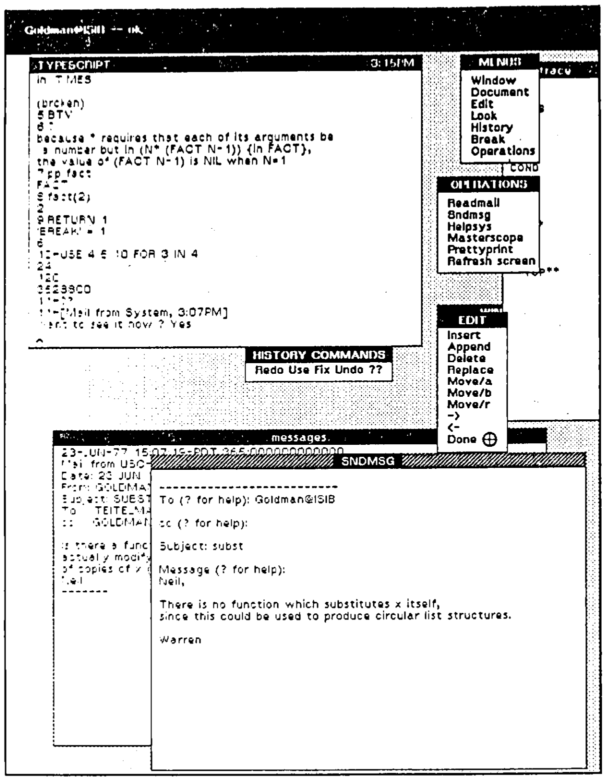
```
TYPESCRIPT                                         3:12PM
in ITIMES

(broken)
5:BTV
6:?
because * requires that each of its arguments be
 a number but in (N* (FACT N-1)) {in FACT},
the value of (FACT N-1) is NIL when N=1
7:pp fact
FACT
8:fact(2)
2
9:RETURN 1
'BREAK' = 1
6
 10←USE 4 5 10 FOR 3 IN 4
24
 120
3628800
 11←??
 11←[Mail from System, 3:07PM]
Want to see it now ? Yes
^
```

```
messages
23-JUN-77 15:07:19-PDT,365;000000000000
Mail from USC-
Date: 23 JUN
From: GOLDMAN
Subject: SUBST
To:   TEITELMA
cc:   GOLDMAN

Is there a func
actually modify
of copies of x
Neil
-------
```

```
SNDMSG
---------------------------
To (? for help): Goldman@ISIB

cc (? for help): subst

Subject: Neil,


Message (? for help):
There is no function which substitutes x itself,
since this could be used to produce circular list structures.

Warren
```

Figure 20

21. Using the EDIT menu, I edit the message, and move "subst" to the subject field, and "Neil," into the body of the message. Then I select the DONE command. The PROMPT window informs me the message has been sent.† Note that I can send a message, change a part of it, and resend it, e.g., to different recipients, again and again.



```
in TIMES

(broken)
5 BTV
6 ?
because * requires that each of its arguments be
 a number but in (N* (FACT N-1)) {in FACT},
the value of (FACT N-1) is NIL when N=1
7 pp fact
FACT
8 fact(2)
2
9 RETURN 1
'BREAK' = 1
6
10-USE 4 5 10 FOR 3 IN 4
24
120
3628800
11-??
11-[Mail from System, 3:07PM]
Want to see it now ? Yes
^
```

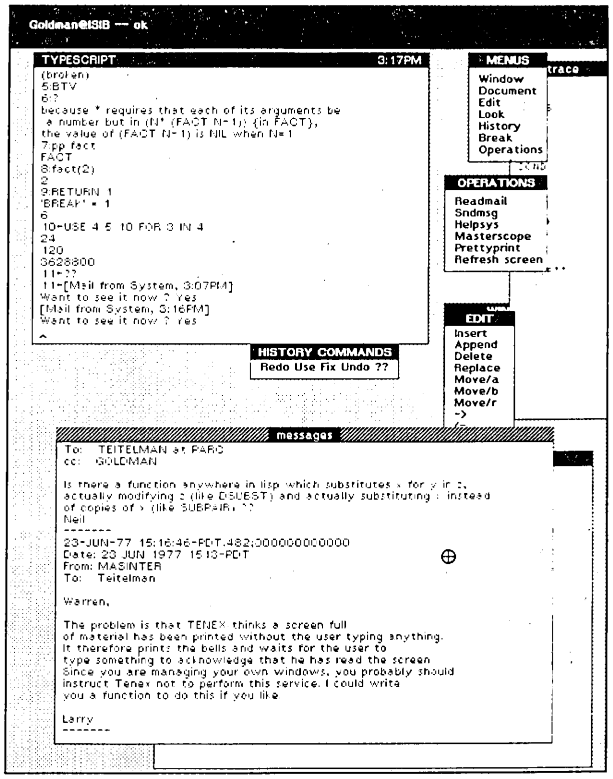
```
23-JUN-77 15:07:19-PDT,365;000000000000
Mail from USC-
Date: 23 JUN
From: GOLDMAN
Subject: SUBST
To:   TEITELMA
cc:   GOLDMAN

Is there a func
actually modify
of copies of x
Neil
-------
```

```
----------------------------
To (? for help): Goldman@ISIB

cc (? for help):

Subject: subst

Message (? for help):
Neil,

There is no function which substitutes x itself,
since this could be used to produce circular list structures.

Warren
```

Figure 21

22. The p.a. now tells me that I have more mail, which I read. It is the response from Masinter. He explains that the bells in my history window are a Tenex feature, and offers to write a Lisp function for me which will turn this Tenex feature off for my specialized application.

†The operation to be performed when the DONE command is selected is specified by the program that originally prints the corresponding material. In the case of SNDMSG, the operation is to send the message. In the case of prettyprint (as illustrated in Figure 13), the operation is to redefine the corresponding function.

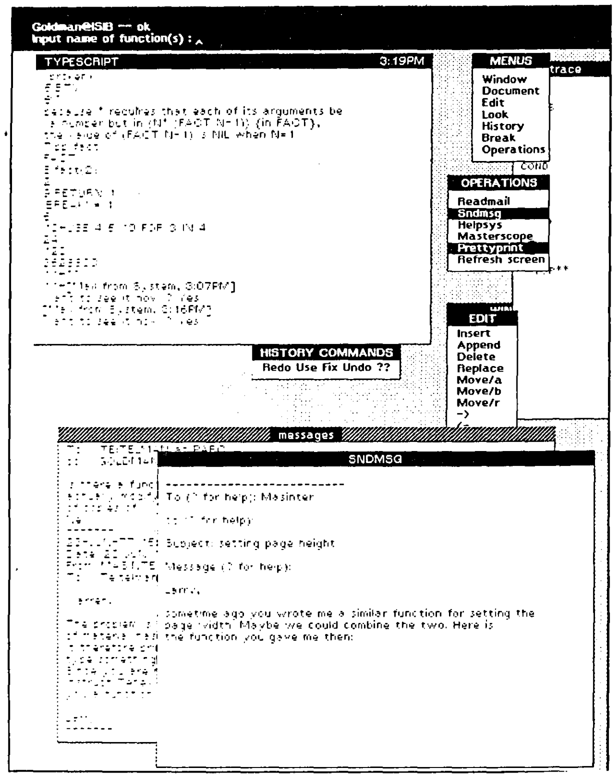


```
(broken)
5:BTV
6:?
because * requires that each of its arguments be
 a number but in (N* (FACT N-1)) {in FACT},
the value of (FACT N-1) is NIL when N=1
7:pp fact
FACT
8:fact(2)
2
9:RETURN 1
'BREAK' = 1
6
10-USE 4 5 10 FOR 3 IN 4
24
120
3628800
11-??
11-[Mail from System, 3:07PM]
Want to see it now ? Yes
[Mail from System, 3:16PM]
Want to see it now ? Yes
^
```

```
To:   TEITELMAN at PARC
cc:   GOLDMAN

Is there a function anywhere in lisp which substitutes x for y in z,
actually modifying z (like DSUBST) and actually substituting x instead
of copies of x (like SUBPAIR) ??
Neil
-------
23-JUN-77 15:16:46-PDT,482;000000000000
Date: 23 JUN 1977 1513-PDT
From: MASINTER
To:   Teitelman

Warren,

The problem is that TENEX thinks a screen full
of material has been printed without the user typing anything.
It therefore prints the bells and waits for the user to
type something to acknowledge that he has read the screen
Since you are managing your own windows, you probably should
instruct Tenex not to perform this service. I could write
you a function to do this if you like.

Larry
-------
```

Figure 22

23. Since this function basically sets the page height, and I already have a function for setting the page *width*, I ask Masinter whether he can simply combine the two operations in a single function. I want to include the definition of the function that sets the page width in the message I send to him. So I type the first part of the message as shown, and then I select the PRETTYPRINT command of the OPERATIONS menu. The PROMPT window asks me to supply the name of the function(s) I want prettyprinted. Figure 23 shows the display as of this point. Note that the caret is in the PROMPT window.



```
Goldman@ISIB -- ok
Input name of function(s) : ^
```

```
because * requires that each of its arguments be
 a number but in (N* (FACT N-1)) {in FACT},
the value of (FACT N-1) is NIL when N=1
```

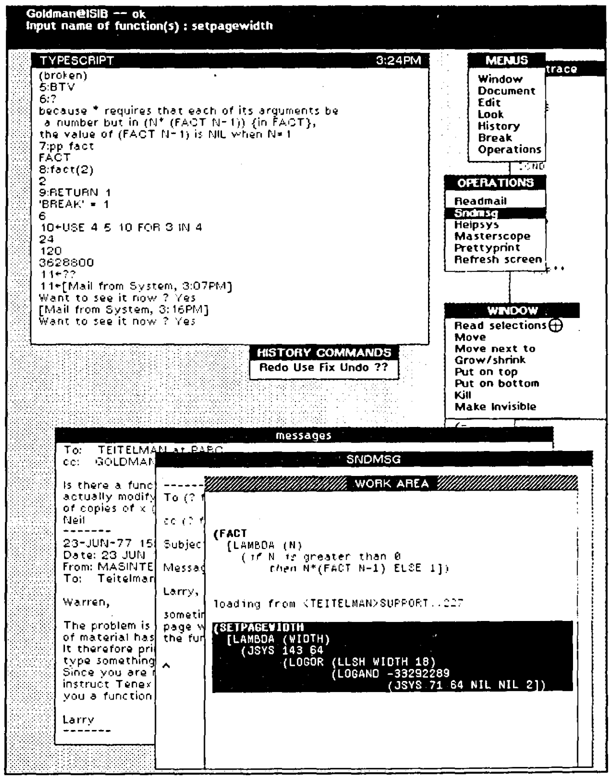
```
[Mail from System, 3:07PM]
Want to see it now ? Yes
[Mail from System, 3:16PM]
Want to see it now ? Yes
```

```
--------------------------
To (? for help): Masinter

cc (? for help):

Subject: setting page height

Message (? for help):

Larry,

Sometime ago you wrote me a similar function for setting the
page width. Maybe we could combine the two. Here is
the function you gave me then:
```

Figure 23

24. I type in the name of my function, SETPAGEWIDTH. The WORK AREA window, which had become covered by the HISTORY, SNDMSG, and MESSAGES window, reappears on top, and the definition of SETPAGEWIDTH is prettyprinted.† I will include this definition in the message which I am composing by using the READ SELECTIONS command in the WINDOW menu. I select the definition, and move the mouse to the READ SELECTIONS command, as shown in Figure 24.

†In this case, the function SETPAGEWIDTH was compiled, and its symbolic definition not loaded into my system. PRETTYPRINT therefore asked the Interlisp file package where the symbolic definition for that function was located, and then loaded it in. All of this happens automatically without any need for user intervention.



```
Goldman@ISIB -- ok
Input name of function(s) : setpagewidth
```

```
(broken)
5:BTV
6:?
because * requires that each of its arguments be
 a number but in (N* (FACT N-1)) {in FACT},
the value of (FACT N-1) is NIL when N=1
7:pp fact
FACT
8:fact(2)
2
9:RETURN 1
'BREAK' = 1
6
10←USE 4 5 10 FOR 3 IN 4
24
120
3628800
11←??
11←[Mail from System, 3:07PM]
Want to see it now ? Yes
[Mail from System, 3:16PM]
Want to see it now ? Yes
```

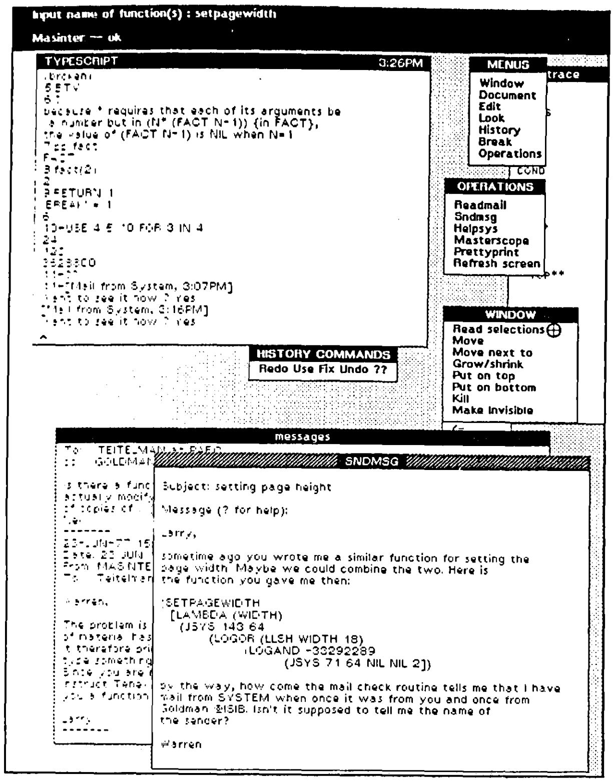
```
To:   TEITELMAN at PARC
cc:   GOLDMAN

Is there a func
actually modify
of copies of x
Neil
-------
23-JUN-77 15
Date: 23 JUN
From: MASINTE
To:   Teitelman

Warren,

The problem is
of material has
It therefore pri
type something
Since you are
instruct Tenex
you a function

Larry
-------
```

```
------
To (?

cc (?

Subjec

Messa

Larry,

someti
page w
the fun
^
```

```
(FACT
  [LAMBDA (N)
    (if N is greater than 0
        then N*(FACT N-1) ELSE 1])

loading from <TEITELMAN>SUPPORT..227

(SETPAGEWIDTH
  [LAMBDA (WIDTH)
    (JSYS 143 64
          (LOGOR (LLSH WIDTH 18)
                 (LOGAND -33292289
                         (JSYS 71 64 NIL NIL 2])
```

Figure 24

25. I select the READ SELECTIONS command, and the effect is the same as though I had typed in the selected material: my SNDMSG window comes back on top, and the definition for SETPAGEWIDTH is inserted into the message, as shown in Figure 25. I complete the message by asking Masinter about a totally different matter, which is why the mail check routine tells me I have mail from SYSTEM, rather than the name of the sender.



```
Input name of function(s) : setpagewidth
Masinter -- ok
```

```
(broken)
5 ETV
6 ?
because * requires that each of its arguments be
a number but in (N* (FACT N-1)) {in FACT},
the value of (FACT N-1) is NIL when N=1
7 ee fact
F--
8 fact(2)
2
9 RETURN 1
BREAK * 1
6
10-USE 4 5 10 FOR 3 IN 4
24
120
3628800
11-??
11-[Mail from System, 3:07PM]
Want to see it now ? yes
[Mail from System, 3:16PM]
Want to see it now ? yes
^
```

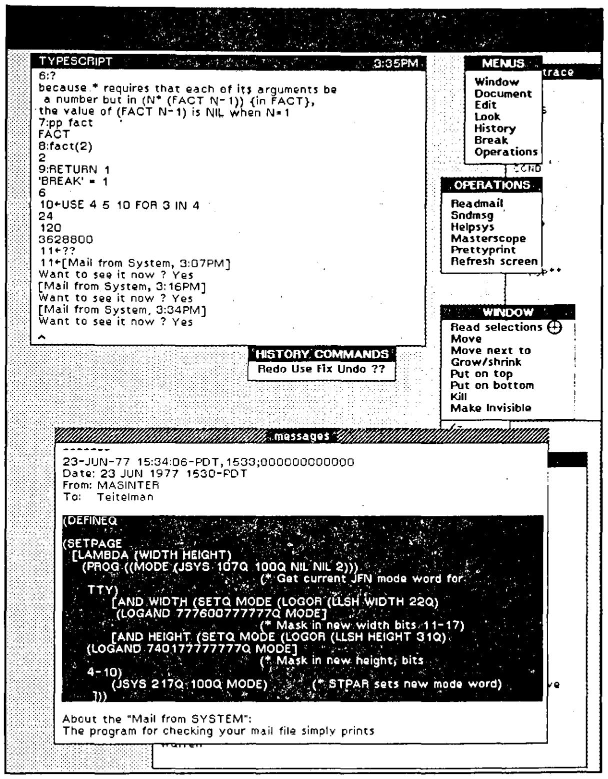
```
To:  TEITELMAN at PARC
cc:  GOLDMAN
```

```
Subject: setting page height

Message (? for help):

Larry,

sometime ago you wrote me a similar function for setting the
page width. Maybe we could combine the two. Here is
the function you gave me then:

(SETPAGEWIDTH
  [LAMBDA (WIDTH)
    (JSYS 143 64
          (LOGOR (LLSH WIDTH 18)
                 (LOGAND -33292289
                         (JSYS 71 64 NIL NIL 2])

by the way, how come the mail check routine tells me that I have
mail from SYSTEM when once it was from you and once from
Goldman @ISIB. Isn't it supposed to tell me the name of
the sender?

Warren
```

Figure 25

This casual exchanging of messages back and forth is an important part of the way that many of us use computers today. Twenty or so of my colleagues are using the same time sharing system as I am, not to mention the much greater number of users on other machines on the ARPA network, and we exchange messages frequently. Thus it is of great value to me to be able to switch contexts from debugging a program to sending or receiving mail with a minimum amount of overhead. In this case, it was particularly important to be able to point at a piece of my programming environment, i.e. the definition of a function, and insert it directly into a message. The same facility would be useful for example in reporting a bug, where I might want to include a sequence of interactions with the system in my message. The inverse operation, of pointing at a piece of a message I receive and installing it in my programming environment, is also very useful, as we will see in the next interaction.

26. The p.a. now tells me I have more mail. It is the reply from Masinter containing the definition for the function SETPAGE. I select the definition, move the mouse to the READ SELECTIONS command in the WINDOW menu, and

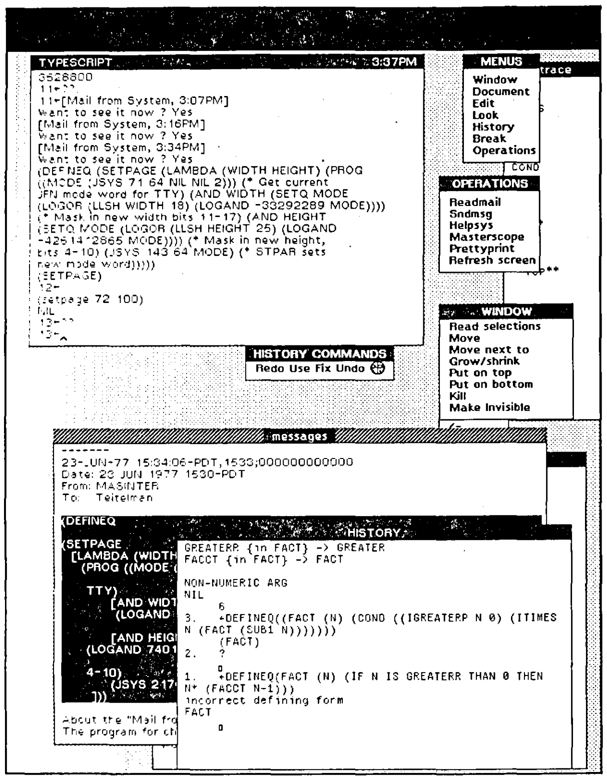


```
TYPESCRIPT                                        3:35PM
6:?
because * requires that each of its arguments be
 a number but in (N* (FACT N-1)) {in FACT},
the value of (FACT N-1) is NIL when N=1
7:pp fact
FACT
8:fact(2)
2
9:RETURN 1
'BREAK' = 1
6
10←USE 4 5 10 FOR 3 IN 4
24
120
3628800
11←??
11←[Mail from System, 3:07PM]
Want to see it now ? Yes
[Mail from System, 3:16PM]
Want to see it now ? Yes
[Mail from System, 3:34PM]
Want to see it now ? Yes

^
```

```
-------
23-JUN-77 15:34:06-PDT,1533;000000000000
Date: 23 JUN 1977 1530-PDT
From: MASINTER
To:   Teitelman

(DEFINEQ

(SETPAGE
  [LAMBDA (WIDTH HEIGHT)
    (PROG ((MODE (JSYS 107Q 100Q NIL NIL 2)))
                                (* Get current JFN mode word for
     TTY)
        [AND WIDTH (SETQ MODE (LOGOR (LLSH WIDTH 22Q)
         (LOGAND 777600777777Q MODE]
                                (* Mask in new width bits 11-17)
        [AND HEIGHT (SETQ MODE (LOGOR (LLSH HEIGHT 31Q)
     (LOGAND 740177777777Q MODE]
                                (* Mask in new height, bits
     4-10)
        (JSYS 217Q 100Q MODE)        (* STPAR sets new mode word)
      ])))

About the "Mail from SYSTEM":
The program for checking your mail file simply prints
```

Figure 26

27. select the READ SELECTIONS command thereby defining SETPAGE, as shown in the TYPESCRIPT window in Figure 27. I now use SETPAGE to set my page height, and then select the ?? command in the HISTORY menu to see if the bells are printed. They aren't: this time the entire history is printed without any pause. (In Figure 27, the HISTORY window shows the end of the history, i.e., the beginning of the session where I defined FACT.)



```
TYPESCRIPT                                        3:37PM
3628800
11←??
11←[Mail from System, 3:07PM]
Want to see it now ? Yes
[Mail from System, 3:16PM]
Want to see it now ? Yes
[Mail from System, 3:34PM]
Want to see it now ? Yes
(DEFINEQ (SETPAGE (LAMBDA (WIDTH HEIGHT) (PROG
((MODE (JSYS 71 64 NIL NIL 2))) (* Get current
JFN mode word for TTY) (AND WIDTH (SETQ MODE
(LOGOR (LLSH WIDTH 18) (LOGAND -33292289 MODE))))
(* Mask in new width bits 11-17) (AND HEIGHT
(SETQ MODE (LOGOR (LLSH HEIGHT 25) (LOGAND
-4251412865 MODE)))) (* Mask in new height,
bits 4-10) (JSYS 143 64 MODE) (* STPAR sets
new mode word)))))
(SETPAGE)
12←
(setpage 72 100)
NIL
13←??
13←^
```

```
-------
23-JUN-77 15:34:06-PDT,1533;000000000000
Date: 23 JUN 1977 1530-PDT
From: MASINTER
To:   Teitelman

(DEFINEQ

(SETPAGE
  [LAMBDA (WIDTH
    (PROG ((MODE (

     TTY)
        [AND WIDT
         (LOGAND

        [AND HEIGI
     (LOGAND 7401

     4-10)
        (JSYS 217
      ]))

About the "Mail fro
The program for ch
```

```
HISTORY
GREATERP {in FACT} -> GREATER
FACCT {in FACT} -> FACT

NON-NUMERIC ARG
NIL
     6
3.   ←DEFINEQ((FACT (N) (COND ((IGREATERP N 0) (ITIMES
N (FACT (SUB1 N)))))))
     (FACT)
2.   ?
     0
1.   ←DEFINEQ(FACT (N) (IF N IS GREATERR THAN 0 THEN
N* (FACCT N-1)))
incorrect defining form
FACT
     0
```

Figure 27

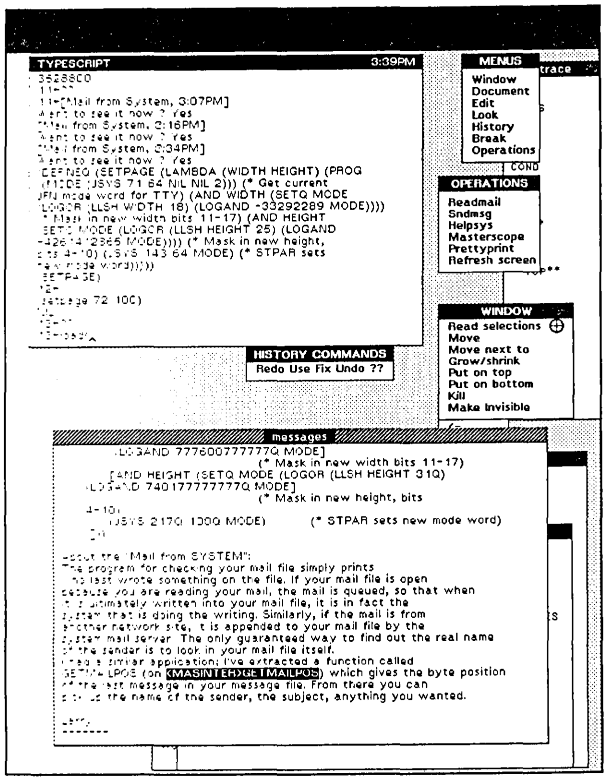
Let us pause now and review the sequence of operations commencing with noticing the problem and culminating in its solution:

1. I noticed a problem,
2. sent a message,
3. received an explanation,
4. sent back a reply containing a piece of one of my programs,
5. received a message containing a program which I could use to fix the problem,
6. installed the program in my environment by pointing at it, and
7. fixed my problem.

This particular problem admittedly was a trivial one, and could easily have been ignored or tolerated by the user. The important point here is that the configuration of the system makes it *so easy for the user to attack and solve such problems that he is willing to do so.* The leverage that the system provides the user is even more valuable when the user is attacking conceptually difficult problems.

28. I use the PUT ON TOP command of the WINDOW menu to bring the message window back on top to read the rest of the message from Masinter. Since the message is too long to fit in the window at one time, I *scroll* the contents of the window to see the rest of the message by placing the mouse in an imaginary bar to the left of the window and pressing the left button (for scrolling up—the right button is used for scrolling down). The line opposite the mouse is then scrolled to the top of the window. Masinter explains that the mail checker I am using simply checks the last user to write on my message file. If my message file is busy, or the mail is coming from over the ARPA network, as was the case with the message from Goldman, then the "user" that actually writes on my message file is, in fact, the system. He suggests that if I want to bother, I can find out the real name of the sender by actually looking in the message file at the message itself. Masinter says he has a function called GETMAILPOS which will return the position of the last message in the file.

I decide to make this change, so I type "LOAD(" to the programmer's assistant (as shown in the TYPESCRIPT window), and then select the name of the file in the message (in order to use the READ SELECTIONS command).

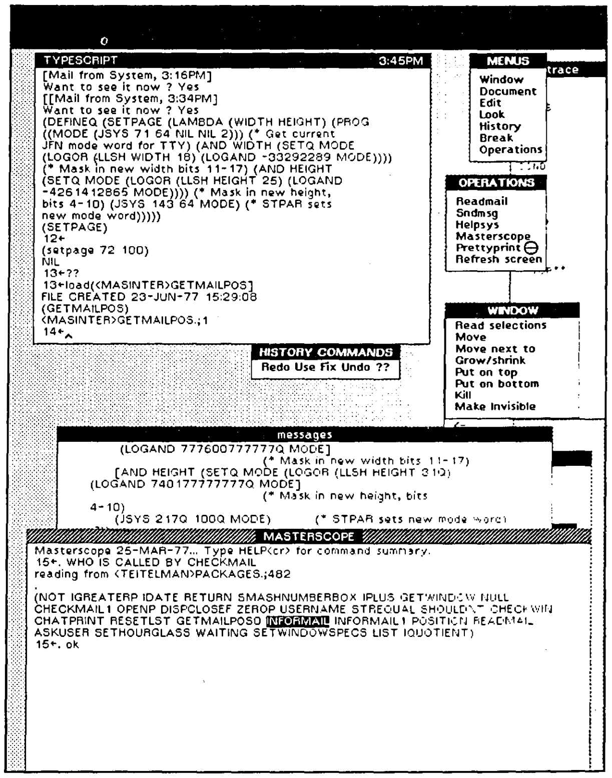


```
3528800
11←^^
11←[Mail from System, 3:07PM]
Want to see it now ? Yes
[Mail from System, 3:16PM]
Want to see it now ? Yes
[Mail from System, 3:34PM]
Want to see it now ? Yes
DEFINEQ (SETPAGE (LAMBDA (WIDTH HEIGHT) (PROG
((MODE (JSYS 71 64 NIL NIL 2))) (* Get current
JFN mode word for TTY) (AND WIDTH (SETQ MODE
(LOGOR (LLSH WIDTH 18) (LOGAND -33292289 MODE))))
(* Mask in new width bits 11-17) (AND HEIGHT
(SETQ MODE (LOGOR (LLSH HEIGHT 25) (LOGAND
-4261412865 MODE)))) (* Mask in new height,
bits 4-10) (JSYS 143 64 MODE) (* STPAR sets
new mode word))))
SETPAGE)
12←
setpage 72 100)
13←
14←^^
15←load(^
```

```
      (LOGAND 777600777777Q MODE]
                                (* Mask in new width bits 11-17)
     [AND HEIGHT (SETQ MODE (LOGOR (LLSH HEIGHT 31Q)
   (LOGAND 740177777777Q MODE]
                                (* Mask in new height, bits
   4-10)
      (JSYS 217Q 100Q MODE)        (* STPAR sets new mode word)
   ]))

About the "Mail from SYSTEM":
The program for checking your mail file simply prints
 who last wrote something on the file. If your mail file is open
because you are reading your mail, the mail is queued, so that when
it is ultimately written into your mail file, it is in fact the
system that is doing the writing. Similarly, if the mail is from
another network site, it is appended to your mail file by the
system mail server. The only guaranteed way to find out the real name
of the sender is to look in your mail file itself.
I had a similar application; I've extracted a function called
GETMAILPOS (on <MASINTER>GETMAILPOS) which gives the byte position
of the last message in your message file. From there you can
pick up the name of the sender, the subject, anything you wanted.

Larry
-------
```

Figure 28

29. I select the READ SELECTIONS command, and the file is loaded, thereby defining the function GETMAILPOS. Now I need to find out where to make the change to inform me of the real identity of the sender. I therefore use the Masterscope command on the OPERATIONS menu to call Masterscope. My interactions with Masterscope are shown in the MASTERSCOPE window at the bottom of the screen in Figure 29. I ask Masterscope the names of all of the functions called by CHECKMAIL. Masterscope obtains and "analyzes" the source definition for CHECKMAIL. I notice the function INFORMAIL among the names of the functions called by CHECKMAIL. INFORMAIL looks like it might be the function I want. I select INFORMAIL and



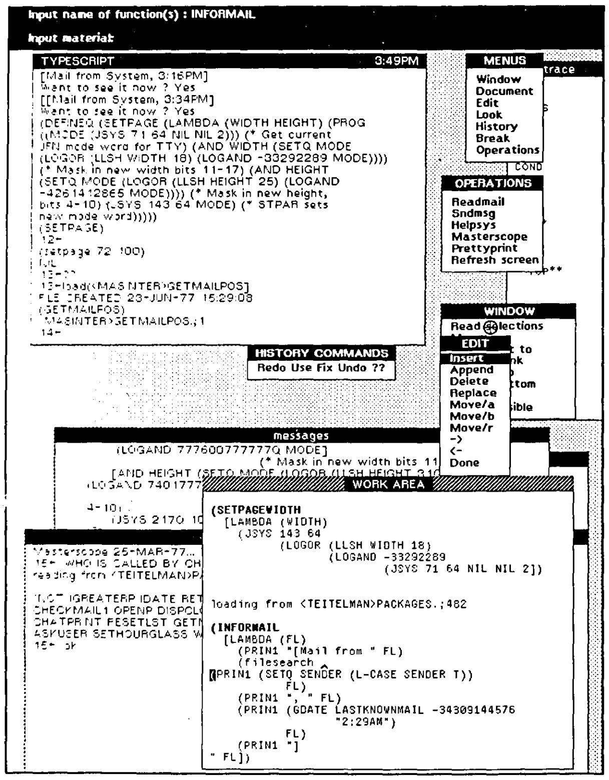
```
TYPESCRIPT                                        3:45PM
[Mail from System, 3:16PM]
Want to see it now ? Yes
[[Mail from System, 3:34PM]
Want to see it now ? Yes
(DEFINEQ (SETPAGE (LAMBDA (WIDTH HEIGHT) (PROG
((MODE (JSYS 71 64 NIL NIL 2))) (* Get current
JFN mode word for TTY) (AND WIDTH (SETQ MODE
(LOGOR (LLSH WIDTH 18) (LOGAND -33292289 MODE))))
(* Mask in new width bits 11-17) (AND HEIGHT
(SETQ MODE (LOGOR (LLSH HEIGHT 25) (LOGAND
-4261412865 MODE)))) (* Mask in new height,
bits 4-10) (JSYS 143 64 MODE) (* STPAR sets
new mode word)))))
(SETPAGE)
 12←
(setpage 72 100)
NIL
 13←??
 13←load(<MASINTER>GETMAILPOS]
FILE CREATED 23-JUN-77 15:29:08
(GETMAILPOS)
<MASINTER>GETMAILPOS.;1
 14←
```

```
messages
      (LOGAND 777600777777Q MODE]
                          (* Mask in new width bits 11-17)
     [AND HEIGHT (SETQ MODE (LOGOR (LLSH HEIGHT 31Q)
   (LOGAND 740177777777Q MODE]
                          (* Mask in new height, bits
   4-10)
     (JSYS 217Q 100Q MODE)      (* STPAR sets new mode word)
```

```
MASTERSCOPE
Masterscope 25-MAR-77... Type HELP<cr> for command summary.
 15←. WHO IS CALLED BY CHECKMAIL
reading from <TEITELMAN>PACKAGES.;482
.
(NOT IGREATERP IDATE RETURN SMASHNUMBERBOX IPLUS GETWINDOW NULL
CHECKMAIL1 OPENP DISPCLOSEF ZEROP USERNAME STREQUAL SHOULDNT CHECKWIN
CHATPRINT RESETLST GETMAILPOSO INFORMAIL INFORMAIL1 POSITION READMAIL
ASKUSER SETHOURGLASS WAITING SETWINDOWSPECS LIST IQUOTIENT)
 15←. ok
```

Figure 29

30. prettyprint it, and see that INFORMAIL is indeed the function that prints the [Mail from --] message, and so is the place to make my modification. I use the INSERT command on the EDIT menu and begin making the change. Figure 30 shows the definition of INFORMAIL with the text "(FILESEARCH" inserted.



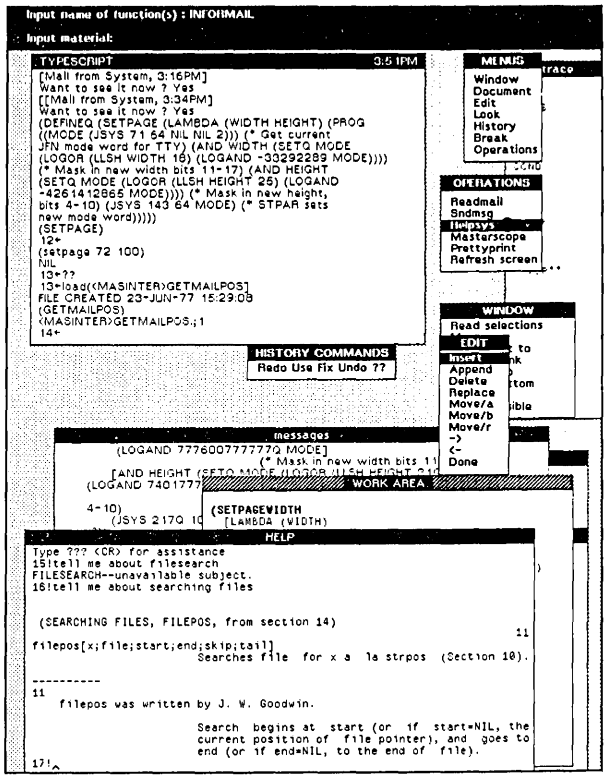
```
Input name of function(s) : INFORMAIL
Input material:
```

```
TYPESCRIPT                                        3:49PM
[Mail from System, 3:16PM]
Want to see it now ? Yes
[[Mail from System, 3:34PM]
Want to see it now ? Yes
(DEFINEQ (SETPAGE (LAMBDA (WIDTH HEIGHT) (PROG
((MODE (JSYS 71 64 NIL NIL 2))) (* Get current
JFN mode word for TTY) (AND WIDTH (SETQ MODE
(LOGOR (LLSH WIDTH 18) (LOGAND -33292289 MODE))))
(* Mask in new width bits 11-17) (AND HEIGHT
(SETQ MODE (LOGOR (LLSH HEIGHT 25) (LOGAND
-4261412865 MODE)))) (* Mask in new height,
bits 4-10) (JSYS 143 64 MODE) (* STPAR sets
new mode word)))))
(SETPAGE)
 12←
(setpage 72 100)
NIL
 13←??
 13←load(<MASINTER>GETMAILPOS]
FILE CREATED 23-JUN-77 15:29:08
(GETMAILPOS)
<MASINTER>GETMAILPOS.;1
 14←
```

```
WORK AREA

(SETPAGEWIDTH
  [LAMBDA (WIDTH)
    (JSYS 143 64
          (LOGOR (LLSH WIDTH 18)
                 (LOGAND -33292289
                         (JSYS 71 64 NIL NIL 2])


loading from <TEITELMAN>PACKAGES.;482

(INFORMAIL
  [LAMBDA (FL)
    (PRIN1 "[Mail from " FL)
    (filesearch
(PRIN1 (SETQ SENDER (L-CASE SENDER T))
            FL)
    (PRIN1 ", " FL)
    (PRIN1 (GDATE LASTKNOWNMAIL -34309144576
                  "2:29AM")
            FL)
    (PRIN1 "]
" FL])
```

Figure 30

31. At this point, I realize that I don't remember how to use the function FILESEARCH, so, *while in the middle of editing*, I use the OPERATIONS menu to call HELPSYS, to interrogate the on-line Interlisp Reference Manual. The interactions with HELPSYS are shown in the HELPSYS window at the bottom of the screen in Figure 31. I first ask HELPSYS about FILESEARCH, and it tells me that there is no such subject in the manual, so I try the phrase "searching files." This causes HELPSYS to give me an explanation of the function FILEPOS, which is the name of the function I want.†



```
[Mail from System, 3:16PM]
Want to see it now ? Yes
[[Mail from System, 3:34PM]
Want to see it now ? Yes
(DEFINEQ (SETPAGE (LAMBDA (WIDTH HEIGHT) (PROG
((MODE (JSYS 71 64 NIL NIL 2))) (* Get current
JFN mode word for TTY) (AND WIDTH (SETQ MODE
(LOGOR (LLSH WIDTH 18) (LOGAND -33292289 MODE))))
(* Mask in new width bits 11-17) (AND HEIGHT
(SETQ MODE (LOGOR (LLSH HEIGHT 25) (LOGAND
-4261412865 MODE)))) (* Mask in new height,
bits 4-10) (JSYS 143 64 MODE) (* STPAR sets
new mode word)))))
(SETPAGE)
 12←
(setpage 72 100)
NIL
 13←??
 13←load(<MASINTER>GETMAILPOS]
FILE CREATED 23-JUN-77 15:29:08
(GETMAILPOS)
<MASINTER>GETMAILPOS.;1
 14←
```

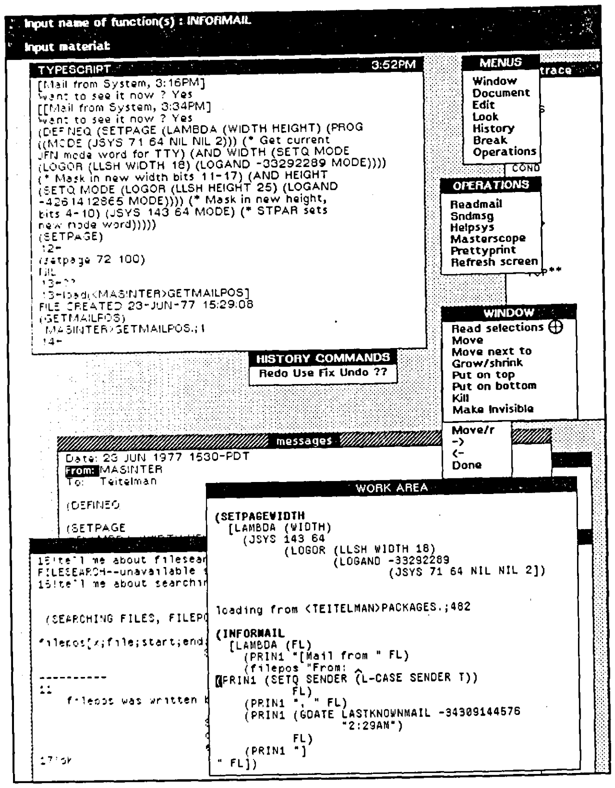
```
        (LOGAND 777600777777Q MODE]
                         (* Mask in new width bits 11
       [AND HEIGHT
    (LOGAND 7401777
    4-10)
        (JSYS 217Q 1
```

```
(SETPAGEWIDTH
  [LAMBDA (WIDTH)
```

```
Type ??? <CR> for assistance
15!tell me about filesearch
FILESEARCH--unavailable subject.
16!tell me about searching files


 (SEARCHING FILES, FILEPOS, from section 14)
                                                          11
filepos[x;file;start;end;skip;tail]
                         Searches file  for x a  la strpos  (Section 10).

----------
11
   filepos was written by J. W. Goodwin.

                         Search  begins at  start (or  if  start=NIL, the
                         current position of  file pointer), and  goes to
                         end (or if end=NIL, to the end of  file).
17!^
```

Figure 31

32. I exit HELPSYS and the WORK AREA window comes back on top, and I am right back in the middle of my edit. I type a sufficient number of backspaces to erase the "SEARCH" in FILESEARCH, and then type POS and continue with my INSERT. The text from the manual about FILEPOS told me that its first argument is the target of the search, in my case the string "From: " in the message. To guarantee that I have the right string, I scroll the MESSAGES window backwards until the beginning of a message is visible, then select this string from an actual message, and then use the READ SELECTIONS command to insert it into my edit.



```
[Mail from System, 3:16PM]
Want to see it now ? Yes
[[Mail from System, 3:34PM]
Want to see it now ? Yes
(DEFINEQ (SETPAGE (LAMBDA (WIDTH HEIGHT) (PROG
((MODE (JSYS 71 64 NIL NIL 2))) (* Get current
JFN mode word for TTY) (AND WIDTH (SETQ MODE
(LOGOR (LLSH WIDTH 18) (LOGAND -33292289 MODE))))
(* Mask in new width bits 11-17) (AND HEIGHT
(SETQ MODE (LOGOR (LLSH HEIGHT 25) (LOGAND
-4261412865 MODE)))) (* Mask in new height,
bits 4-10) (JSYS 143 64 MODE) (* STPAR sets
new mode word)))))
(SETPAGE)
 12←
(setpage 72 100)
NIL
 13←??
 13←load(<MASINTER>GETMAILPOS]
FILE CREATED 23-JUN-77 15:29:08
(GETMAILPOS)
<MASINTER>GETMAILPOS.;1
 14←
```

```
Date: 23 JUN 1977 1530-PDT
From: MASINTER
To:   Teitelman

(DEFINEQ

(SETPAGE
```

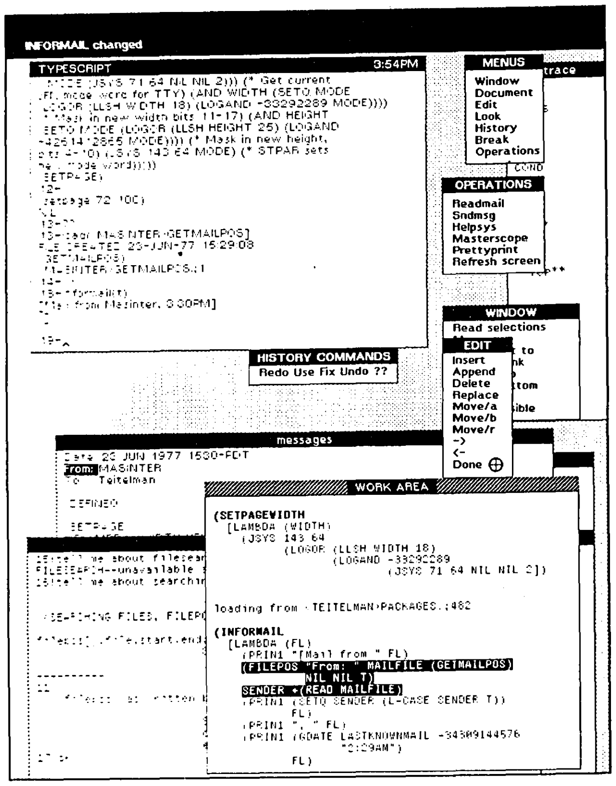
```
15!tell me about filesear
FILESEARCH--unavailable
15!tell me about searchi

 (SEARCHING FILES, FILEP

filepos[x;file;start;end;

----------
11
   filepos was written b

17!ok
```

```
(SETPAGEWIDTH
  [LAMBDA (WIDTH)
    (JSYS 143 64
          (LOGOR (LLSH WIDTH 18)
                 (LOGAND -33292289
                         (JSYS 71 64 NIL NIL 2])


loading from <TEITELMAN>PACKAGES.;482

(INFORMAIL
  [LAMBDA (FL)
    (PRIN1 "[Mail from " FL)
    (filepos "From: 
(PRIN1 (SETQ SENDER (L-CASE SENDER T))
            FL)
    (PRIN1 ", " FL)
    (PRIN1 (GDATE LASTKNOWNMAIL -34309144576
                  "2:29AM")
            FL)
    (PRIN1 "]
" FL])
```

Figure 32

†If that had failed, I would have asked HELPSYS about FILES, which would have given me a list of all words or topics beginning with the letters FILE, just as though I had looked in the index of the manual itself.

33. I complete my INSERT and select the DONE command. The PROMPT window says that the function INFORMAIL has been changed. Basically, the change I made to INFORMAIL says to begin searching the mailfile as of the location specified by Masinter's function GETMAILPOS, looking for the string "From: ",† and then to read a single word from the file and set SENDER to this word. I test out my change by typing INFORMAIL(T). The last time I got mail INFORMAIL said [Mail from System]. This time it tells me Mail from Masinter, so the change worked.



```
MODE (JSYS 71 64 NIL NIL 2))) (* Get current
JFN mode word for TTY) (AND WIDTH (SETQ MODE
(LOGOR (LLSH WIDTH 18) (LOGAND -33292289 MODE))))
(* Mask in new width bits 11-17) (AND HEIGHT
(SETQ MODE (LOGOR (LLSH HEIGHT 25) (LOGAND
-42614 2865 MODE)))) (* Mask in new height,
bits 4-10) (JSYS 143 64 MODE) (* STPAR sets
the mode word))))
SETPAGE)
12←
setpage 72 100)
NL
13←??
13←load MASINTER>GETMAILPOS]
FILE CREATED 23-JUN-77 15:29:08
GETMAILPOS)
MASINTER>GETMAILPOS;1
14←
15←informail(t)
[Mail from Masinter, 3:30PM]

19←
```

```
Date 23 JUN 1977 1530-PDT
From: MASINTER
To: Teitelman
```

```
(SETPAGEWIDTH
  [LAMBDA (WIDTH)
    (JSYS 143 64
          (LOGOR (LLSH WIDTH 18)
                 (LOGAND -33292289
                         (JSYS 71 64 NIL NIL 2])


loading from <TEITELMAN>PACKAGES.;482

(INFORMAIL
  [LAMBDA (FL)
    (PRIN1 "[Mail from " FL)
    (FILEPOS "From: " MAILFILE (GETMAILPOS)
             NIL NIL T)
    SENDER ←(READ MAILFILE)
    (PRIN1 (SETQ SENDER (L-CASE SENDER T))
           FL)
    (PRIN1 ", " FL)
    (PRIN1 (GDATE LASTKNOWNMAIL -34309144576
                  "3:29AM")
           FL)
```

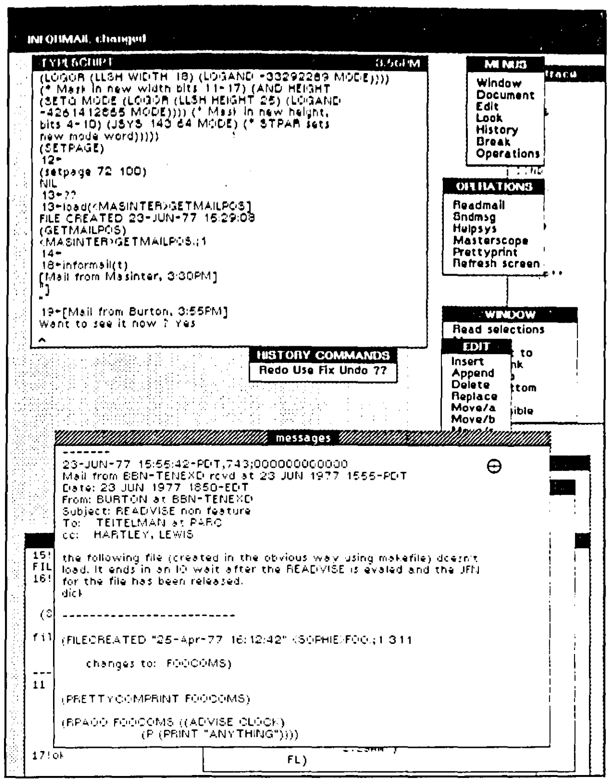
Figure 33

Let us again review the sequence of operations:

1. I observed some undesirable behaviour in a program I was using;
2. sent a message inquiring about the behaviour;
3. got a reply back suggesting the nature of the problem, how it might be changed, and a program which would help in making the change;
4. used Masterscope to find out what to change
5. began making the change and then in the middle,
6. used Helpsys to tell me how to make the change,
7. completed the change, and tested it successfully.

†The extra arguments to FILEPOS specify that the search is to stop *after* the string, not at its beginning as is the default case.

34. The p.a. (via INFORMAIL) now tells me that I have mail from Burton, which I agree to see. However, I realize that I would like INFORMAIL to say I had mail from Burton at BBN-TENEXD, just as it does in the message file, rather than just Burton.

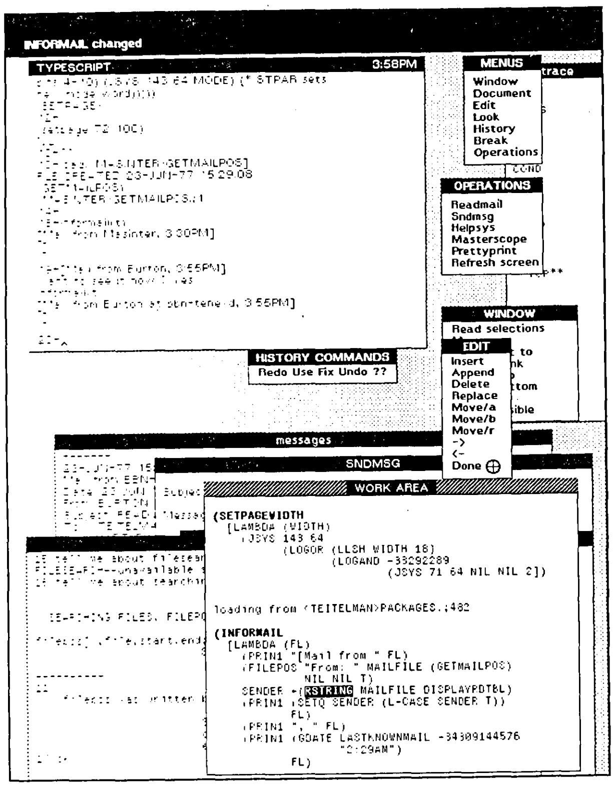


```
(LOGOR (LLSH WIDTH 18) (LOGAND -33292289 MODE))))
(* Mask in new width bits 11-17) (AND HEIGHT
(SETQ MODE (LOGOR (LLSH HEIGHT 25) (LOGAND
-42614 12865 MODE)))) (* Mask in new height,
bits 4-10) (JSYS 143 64 MODE) (* STPAR sets
new mode word)))))
(SETPAGE)
 12←
(setpage 72 100)
NIL
 13←??
 13←load(<MASINTER>GETMAILPOS]
FILE CREATED 23-JUN-77 15:29:08
(GETMAILPOS)
<MASINTER>GETMAILPOS.;1
 14←
 18←informail(t)
[Mail from Masinter, 3:30PM]
"]

 19←[Mail from Burton, 3:55PM]
Want to see it now ? Yes
^
```

```
-------
23-JUN-77 15:55:42-PDT,743;000000000000
Mail from BBN-TENEXD rcvd at 23 JUN 1977 1555-PDT
Date: 23 JUN 1977 1850-EDT
From: BURTON at BBN-TENEXD
Subject: READVISE non feature
To:   TEITELMAN at PARC
cc:   HARTLEY, LEWIS

the following file (created in the obvious way using makefile) doesn't
load. It ends in an IO wait after the READVISE is evaled and the JFN
for the file has been released.
dick

------------------------

(FILECREATED "25-Apr-77 16:12:42" <SOPHIE>FOO.;1 311

    changes to:  FOOCOMS)


(PRETTYCOMPRINT FOOCOMS)

(RPAQQ FOOCOMS ((ADVISE CLOCK)
        (P (PRINT "ANYTHING"))))
```

Figure 34

35. The problem is that the Interlisp function READ, which I used in the change to INFORMAIL, just returns the next expression/word in the file, which in this case was simply Burton. I should have used the function RSTRING, which will read everything up to the next carriage return. Therefore, I simply bring my WORK AREA window back on top, and edit the definition of INFORMAIL, replacing the call to READ by an appropriate call to RSTRING. I then select the DONE command. The PROMPT window tells me me that INFORMAIL has been changed (again). I test out the change by typing INFORMAIL(T). This time INFORMAIL tells me I have mail from Burton at bbn-tenexd, exactly as I planned.

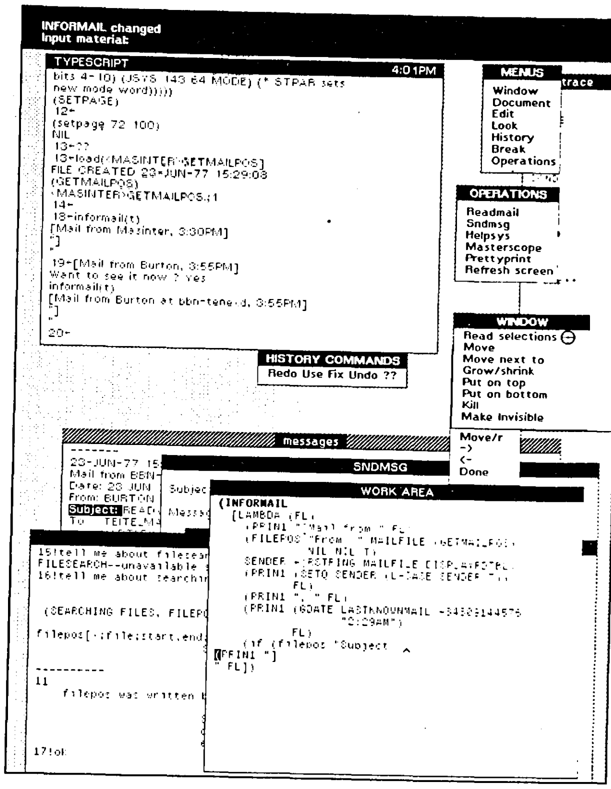


```
[Mail from Burton at bbn-tenexd, 3:55PM]
```

```
(SETPAGEWIDTH
  [LAMBDA (WIDTH)
    (JSYS 143 64
          (LOGOR (LLSH WIDTH 18)
                 (LOGAND -33292289
                         (JSYS 71 64 NIL NIL 2])


loading from <TEITELMAN>PACKAGES.;482

(INFORMAIL
  [LAMBDA (FL)
    (PRIN1 "[Mail from " FL)
    (FILEPOS "From: " MAILFILE (GETMAILPOS)
             NIL NIL T)
    SENDER ←(RSTRING MAILFILE DISPLAYRDTBL)
    (PRIN1 (SETQ SENDER (L-CASE SENDER T))
           FL)
    (PRIN1 ", " FL)
    (PRIN1 (GDATE LASTKNOWNMAIL -34309144576
                  "2:29AM")
           FL)
```

Figure 35

36. As long as I am at it, I decide I would also like INFORMAIL to tell me the subject of the message, so I edit INFORMAIL to search the mailfile for the string "Subject:", which I again obtain from a message itself via the READ SELECTIONS command. Figure 36 shows the edit as of this point.



```
INFORMAIL changed
Input material:
```

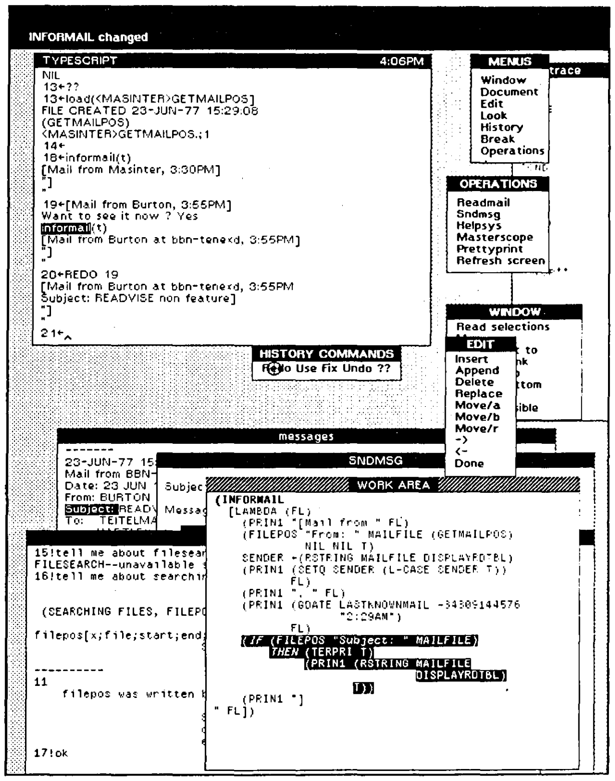
```
TYPESCRIPT                                   4:01PM
bits 4-10) (JSYS 143 64 MODE) (* STPAR sets
new mode word))))
(SETPAGE)
 12←
(setpage 72 100)
NIL
 13←??
 13←load(<MASINTER>GETMAILPOS]
FILE CREATED 23-JUN-77 15:29:08
(GETMAILPOS)
<MASINTER>GETMAILPOS.;1
 14←
 18←informail(t)
[Mail from Masinter, 3:30PM]
"]
"
 19←[Mail from Burton, 3:55PM]
Want to see it now ? Yes
informail(t)
[Mail from Burton at bbn-tenexd, 3:55PM]
"]
"
20←
```

```
-------
23-JUN-77 15
Mail from BBN-
Date: 23 JUN
From: BURTON
Subject: READ
To:   TEITELMA
```

```
15!tell me about filesear
FILESEARCH--unavailable
16!tell me about searchin

 (SEARCHING FILES, FILEP

filepos[x;file;start,end;

----------
11
    filepos was written b

17!ok
```

```
(INFORMAIL
  [LAMBDA (FL)
    (PRIN1 "[Mail from " FL)
    (FILEPOS "From: " MAILFILE (GETMAILPOS)
             NIL NIL T)
    SENDER ←(RSTRING MAILFILE DISPLAYRDTBL)
    (PRIN1 (SETQ SENDER (L-CASE SENDER T))
           FL)
    (PRIN1 ", " FL)
    (PRIN1 (GDATE LASTKNOWNMAIL -34309144576
                  "2:29AM")
           FL)
    (if (filepos "Subject
(PRIN1 "]
" FL])
```

37. I complete the edit, which basically says that if the string "Subject:" is found in the message, INFORMAIL should print it and the rest of the line that follows it. I select the DONE command, and the PROMPT window reports that INFORMAIL has been changed. I test out my change, this time by selecting the previous event in the TYPESCRIPT window and then using the REDO command on the HISTORY menu. As shown in Figure 37, INFORMAIL tells me the full name of the sender, plus the subject.



```
INFORMAIL changed
```

```
TYPESCRIPT                                   4:06PM
NIL
 13←??
 13←load(<MASINTER>GETMAILPOS]
FILE CREATED 23-JUN-77 15:29:08
(GETMAILPOS)
<MASINTER>GETMAILPOS.;1
 14←
 18←informail(t)
[Mail from Masinter, 3:30PM]
"]
"
 19←[Mail from Burton, 3:55PM]
Want to see it now ? Yes
informail(t)
[Mail from Burton at bbn-tenexd, 3:55PM]
"]
"
20←REDO 19
[Mail from Burton at bbn-tenexd, 3:55PM
Subject: READVISE non feature]
"]
"
21←
```

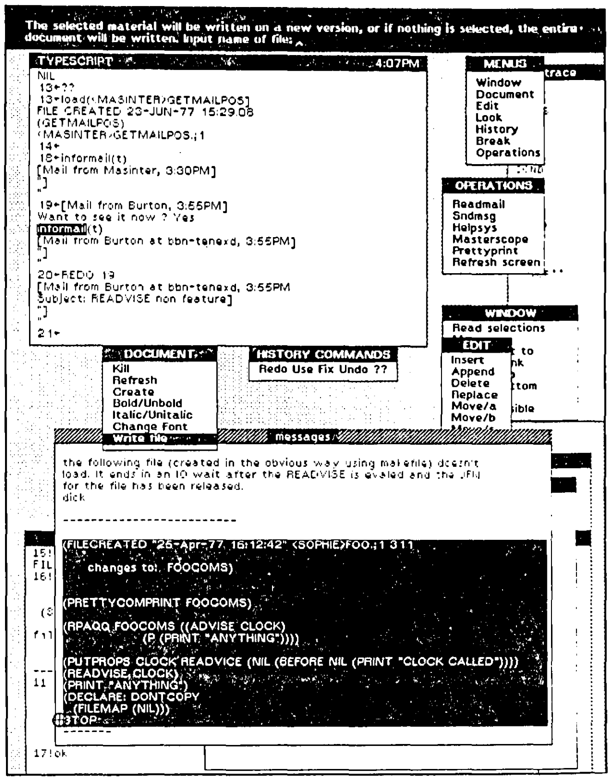
```
-------
23-JUN-77 15
Mail from BBN-
Date: 23 JUN
From: BURTON
Subject: READV
To:   TEITELMA
```

```
15!tell me about filesear
FILESEARCH--unavailable
16!tell me about searchin

 (SEARCHING FILES, FILEP

filepos[x;file;start;end;

----------
11
    filepos was written b

17!ok
```

```
(INFORMAIL
  [LAMBDA (FL)
    (PRIN1 "[Mail from " FL)
    (FILEPOS "From: " MAILFILE (GETMAILPOS)
             NIL NIL T)
    SENDER ←(RSTRING MAILFILE DISPLAYRDTBL)
    (PRIN1 (SETQ SENDER (L-CASE SENDER T))
           FL)
    (PRIN1 ", " FL)
    (PRIN1 (GDATE LASTKNOWNMAIL -34309144576
                  "2:29AM")
           FL)
    (IF (FILEPOS "Subject: " MAILFILE)
        THEN (TERPRI T)
             (PRIN1 (RSTRING MAILFILE
                             DISPLAYRDTBL)
                    T))
    (PRIN1 "]
" FL])
```

Figure 37

38. I now read the message from Burton, which describes a short file that he says will not load correctly. In order to check this out, I need to make such a file and load it and see why it fails. I bring up the DOCUMENT menu and select the WRITE command. The PROMPT window tells me that I should select the material I want written onto the file, and asks me to supply the name of the file. I select the corresponding portion of my message. Figure 38 shows the display at this point.



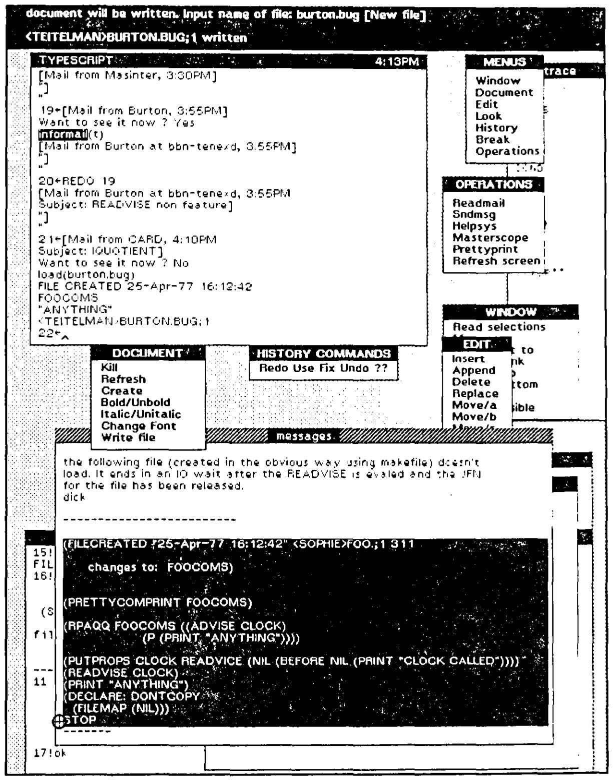
```
The selected material will be written on a new version, or if nothing is selected, the entire
document will be written. Input name of file: ^
```

```
TYPESCRIPT                                        4:07PM
NIL
 13←??
 13←load(<MASINTER>GETMAILPOS]
FILE CREATED 23-JUN-77 15:29.08
(GETMAILPOS)
<MASINTER>GETMAILPOS.;1
 14←
 18←informail(t)
[Mail from Masinter, 3:30PM]
"]
"

 19←[Mail from Burton, 3:55PM]
Want to see it now ? Yes
informail(t)
[Mail from Burton at bbn-tenexd, 3:55PM]
"]
"

20←REDO 19
[Mail from Burton at bbn-tenexd, 3:55PM
Subject: READVISE non feature]
"]
"

21←
```

```
messages
the following file (created in the obvious way using makefile) doesn't
load. It ends in an IO wait after the READVISE is evaled and the JFN
for the file has been released.
dick

-------------------------

(FILECREATED "25-Apr-77 16:12:42" <SOPHIE>FOO.;1 311

     changes to: FOOCOMS)

(PRETTYCOMPRINT FOOCOMS)

(RPAQQ FOOCOMS ((ADVISE CLOCK)
           (P (PRINT "ANYTHING"))))

(PUTPROPS CLOCK READVICE (NIL (BEFORE NIL (PRINT "CLOCK CALLED"))))
(READVISE CLOCK)
(PRINT "ANYTHING")
(DECLARE: DONTCOPY
  (FILEMAP (NIL)))
STOP
-------
```

Figure 38

39. I give the name of the file to be created, BURTON.BUG, and the PROMPT window tells me that the file has been written. At this point, the programmer's assistant tells me I have a message from Card. Since I am in the middle of something, I decide *not* to read the message now, and type No to the question "Want to see it now ?". I load the file BURTON.BUG I just created, and it loads successfully.



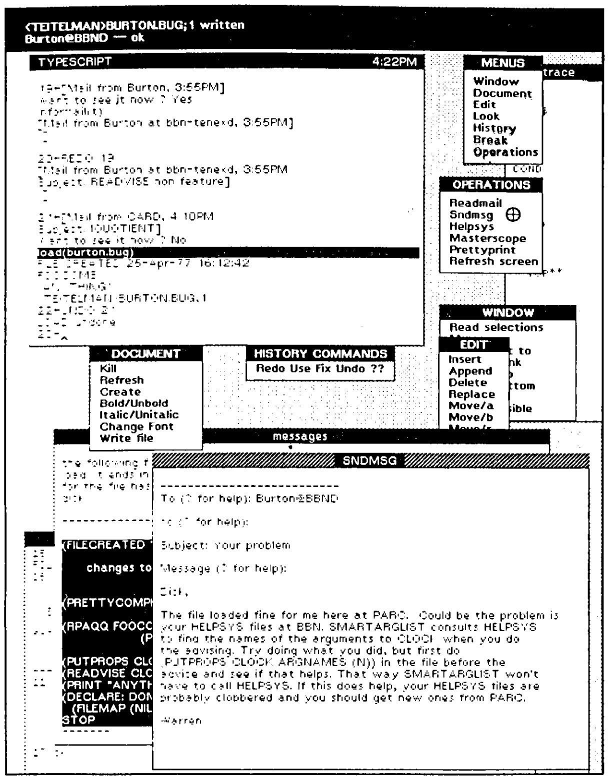
```
document will be written. Input name of file: burton.bug [New file]
<TEITELMAN>BURTON.BUG;1 written
```

```
TYPESCRIPT                                        4:13PM
[Mail from Masinter, 3:30PM]
"]
"

 19←[Mail from Burton, 3:55PM]
Want to see it now ? Yes
informail(t)
[Mail from Burton at bbn-tenexd, 3:55PM]
"]
"

20←REDO 19
[Mail from Burton at bbn-tenexd, 3:55PM
Subject: READVISE non feature]
"]
"

21←[Mail from CARD, 4:10PM
Subject: IQUOTIENT]
Want to see it now ? No
load(burton.bug)
FILE CREATED 25-Apr-77 16:12:42
FOOCOMS
"ANYTHING"
<TEITELMAN>BURTON.BUG;1
22←^
```

```
messages
the following file (created in the obvious way using makefile) doesn't
load. It ends in an IO wait after the READVISE is evaled and the JFN
for the file has been released.
dick

-------------------------

(FILECREATED "25-Apr-77 16:12:42" <SOPHIE>FOO.;1 311

     changes to: FOOCOMS)

(PRETTYCOMPRINT FOOCOMS)

(RPAQQ FOOCOMS ((ADVISE CLOCK)
           (P (PRINT "ANYTHING"))))

(PUTPROPS CLOCK READVICE (NIL (BEFORE NIL (PRINT "CLOCK CALLED"))))
(READVISE CLOCK)
(PRINT "ANYTHING")
(DECLARE: DONTCOPY
  (FILEMAP (NIL)))
STOP
-------
```

Figure 39

40. Since the process of loading this file made some changes to my environment, namely advising the function CLOCK, I *undo* this operation by selecting the corresponding event in the TYPESCRIPT window, and then selecting the UNDO command on the HISTORY menu. I then send Burton a message asking for more details, and suggest that the problem may be due to some files having gotten smashed at BBN.



```
<TEITELMAN>BURTON.BUG;1 written
Burton@BBND — ok
```

TYPESCRIPT 4:22PM

```
19←[Mail from Burton, 3:55PM]
Want to see it now ? Yes
(formailit)
[Mail from Burton at bbn-tenexd, 3:55PM]

20←REDO 19
[Mail from Burton at bbn-tenexd, 3:55PM
Subject: READVISE non feature]

21←[Mail from CARD, 4:10PM
Subject: IQUOTIENT]
Want to see it now ? No
load(burton.bug)
FILE CREATED 25-Apr-77 16:12:42
FOOCOMS
"ANYTHING"
TEITELMAN>BURTON.BUG;1
22←UNDO 21
LOAD undone
23←^
```

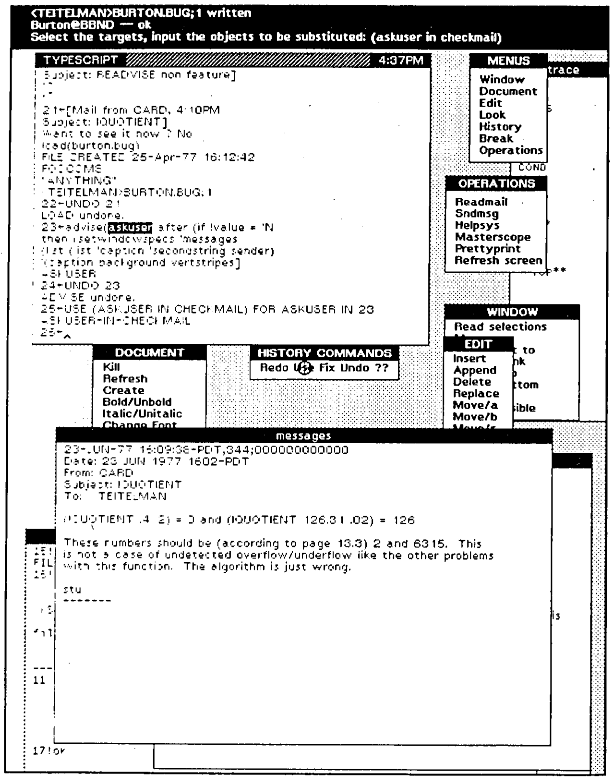
SNDMSG

```
---------------------------
To (? for help): Burton@BBND

cc (? for help):

Subject: Your problem

Message (? for help):

Dick,

The file loaded fine for me here at PARC.  Could be the problem is
your HELPSYS files at BBN. SMARTARGLIST consults HELPSYS
to find the names of the arguments to CLOCK when you do
the advising. Try doing what you did, but first do
(PUTPROPS CLOCK ARGNAMES (N)) in the file before the
advice and see if that helps. That way SMARTARGLIST won't
have to call HELPSYS. If this does help, your HELPSYS files are
probably clobbered and you should get new ones from PARC.

Warren
```

Figure 40

41. I now go back and select the READMAIL command on the OPERATIONS menu to read Card's message, which is a comment about the Interlisp manual, which I will respond to. However, I realize that I could easily have forgotten about Card's message and gone on to something else, so I decide I would like the mail checker to remind me, by changing the caption of my message window, that I have mail waiting when I decline to read it immediately. I will perform this change by simply *advising* the Interlisp function ASKUSER,† which is responsible for the "Want to see it now ? - Yes/No" interaction. I advise ASKUSER AFTER, i.e., the advice will be executed on the way out of the function, if its value is 'N, then to change the caption as indicated. Then I realize that this change will affect *all* calls in the system to ASKUSER, whereas I only want this to happen on calls to ASKUSER from CHECKMAIL. So I select ASKUSER in event 23, i.e., the ADVISE operation, then select the UNDO command on the HISTORY menu to undo this event. Then select the USE command to reexecute the ADVISE operation using (ASKUSER IN CHECKMAIL) instead of ASKUSER. Figure 41 shows the display after the ADVISE operation has been reexecuted.

†Advising is an Interlisp facility which lets the user treat a function, or a particular call to a function, as a black box, and make changes that affect it on entry or exit, without having to be aware of the details of what is inside the box. It is described in [Teil]. Advising is often used for reconfiguring system programs, and also for trying out changes to the user's programs, with minimal investment in order to see how they work, before going back and making the changes in some more permanent fashion.



```
<TEITELMAN>BURTON.BUG;1 written
Burton@BBND — ok
Select the targets, input the objects to be substituted: (askuser in checkmail)
```

TYPESCRIPT 4:37PM

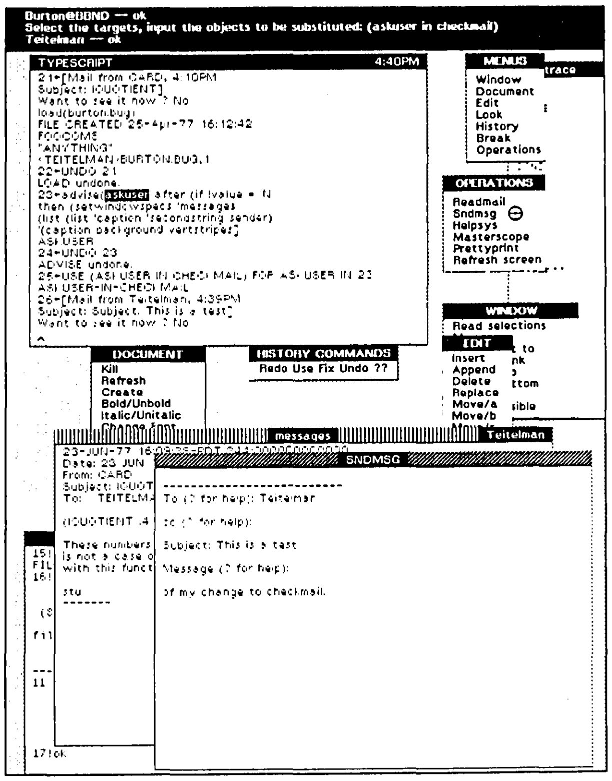
```
Subject: READVISE non feature]

21←[Mail from CARD, 4:10PM
Subject: IQUOTIENT]
Want to see it now ? No
load(burton.bug)
FILE CREATED 25-Apr-77 16:12:42
FOOCOMS
"ANYTHING"
TEITELMAN>BURTON.BUG;1
22←UNDO 21
LOAD undone.
23←advise(askuser after (if !value = 'N
then (setwindowspecs 'messages
(list (list 'caption 'secondstring sender)
'(caption background vertstripes]
ASKUSER
24←UNDO 23
ADVISE undone.
25←USE (ASKUSER IN CHECKMAIL) FOR ASKUSER IN 23
ASKUSER-IN-CHECKMAIL
26←^
```

messages

```
23-JUN-77 16:09:36-PDT,344;000000000000
Date: 23 JUN 1977 1602-PDT
From: CARD
Subject: IQUOTIENT
To:   TEITELMAN

(IQUOTIENT .4 .2) = 0 and (IQUOTIENT 126.31 .02) = 126

These numbers should be (according to page 13.3) 2 and 6315.  This
is not a case of undetected overflow/underflow like the other problems
with this function.  The algorithm is just wrong.

stu
-------
```

Figure 41

42. I test out my change by sending myself a test message, and answering No when asked if I want to see it now. The caption on my MESSAGES window is changed so that the name of the sender of the message, in this case me, appears in the right hand corner of the caption, and the background of the caption is changed to vertical stripes.



```
Burton@BBND — ok
Select the targets, input the objects to be substituted: (askuser in checkmail)
Teitelman — ok
```

```
TYPESCRIPT                                        4:40PM
21←[Mail from CARD, 4:10PM
Subject: IQUOTIENT]
Want to see it now ? No
load(burton.bug)
FILE CREATED 25-Apr-77 16:12:42
FOOCOMS
"ANYTHING"
<TEITELMAN>BURTON.BUG;1
22←UNDO 21
LOAD undone.
23←advise(askuser after (if !value = 'N
then (setwindowspecs 'messages
(list (list 'caption 'secondstring sender)
'(caption background vertstripes]
ASKUSER
24←UNDO 23
ADVISE undone.
25←USE (ASKUSER IN CHECKMAIL) FOR ASKUSER IN 23
ASKUSER-IN-CHECKMAIL
26←[Mail from Teitelman, 4:39PM
Subject: Subject: This is a test]
Want to see it now ? No
^
```

```
23-JUN-77 16:09:35-PDT
Date: 23 JUN
From: CARD
Subject: IQUOT
To:   TEITELMA

(IQUOTIENT .4

These numbers
is not a case o
with this funct

stu
-------
```

```
SNDMSG
---------------------------
To (? for help): Teitelman

cc (? for help):

Subject: This is a test

Message (? for help):

of my change to checkmail.
```

```
15!
FIL
16!
(S
fil
---
11
17!ok
```

Figure 42

Conclusions

The system decribed in this paper has been in use by actual users other than the author only a few months. However, our conjectures about the usefulness of this kind of facility were if anything conservative. The ability to suspend an operation, perform other operations, and then return without loss of context is widely appreciated. The technique of using different windows for different tasks does make this switching of contexts easy and painless. Even when the user is not switching contexts, the use of multiple windows is extremely helpful. For example, a standard complaint with conventional display terminals is that material that the user wants to refer to repeatedly, e.g., a printout of some function, or a record of some complicated interaction, is displaced by subsequent, incidental interactions with the system. In this situation when using a hard copy terminal, the user simply tears off the portion he is interested in and saves it beside his keyboard. Being able to freeze a portion of the user's interactions in a separate window, such as the WORK AREA, while allowing subsequent interactions to scroll off the screen seems to combine some of the best aspects of hardcopy and display terminals.

Finally, users just seem to enjoy aesthetically the style of interacting with the system, such as using menus, the feedback via the prompt window and changing cursors, being able to scroll the windows back and forth, etc. We think this is an area that will see an increasing amount of activity in the future as the cost of bit map displays and the necessary computing power to maintain them continues to drop.

REFERENCES

[Bob] Bobrow, D. G., and Wegbreit B., "A Model and Stack Implementation for Multiple Environments," *Communications of the ACM*, Vol. 16, 10 October 1973.

[Eng] English, W. K., Engelbart, D. C., and Berman, M. L., "Display Selection Techniques for Text Manipulation," *IEEE Transactions on Human Factors in Electronics*, Vol. HFE-8, No. 1, March 1967.

[LRG] Learning Research Group, *Personal Dynamic Media*, Xerox Palo Alto Research Center, 1976. Excerpts published in *IEEE Computer Magazine*, March 1977.

[San] Sandewall, E, "Programming in an Interactive Environment: The Lisp Experience," Matematiska Institutionen, University of Linkoping, Sweden. (to be published in *CACM*).

[Spr] Sproull, R. F., and Thomas, E. L., "A Network Graphics Protocol," *Computer Graphics, SIGGRAPH Quarterly*, Fall 1974.

[Swi] Swinehart, D. C., "Copilot: A Multiple Process Approach to Interactive Programming Systems," Stanford Artificial Intelligence Laboratory Memo AIM-230, Stanford University, July 1974.

[Tei1] Teitelman, W. "Toward a Programming Laboratory," in Walker, D. (ed.) *International Joint Conference on Artificial Intelligence*, May 1969.

[Tei2] Teitelman, W. "Automated Programmering - The Programmer's Assistant," *Proceedings of the Fall Joint Computer Conference*, December 1972.

[Tei3] Teitelman, W. "CLISP - Conversational Lisp," *Third International Joint Conference on Artificial Intelligence*, August 1973.

[Tei4] Teitelman, W. et al., *Interlisp Reference Manual*, Dec. 1975, Xerox Palo Alto Research Center.

XEROX

XEROX A Display Oriented Programmer's Assistant By Warren Teitelman CSL 77-3

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