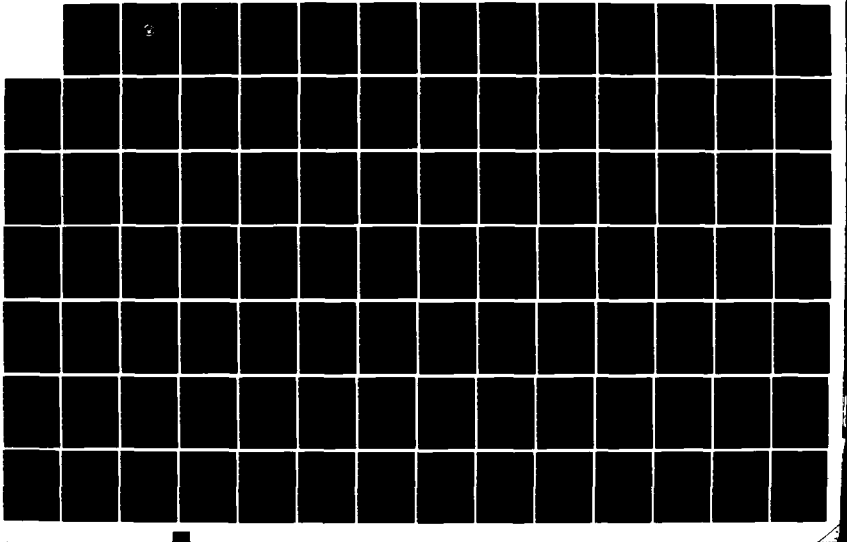
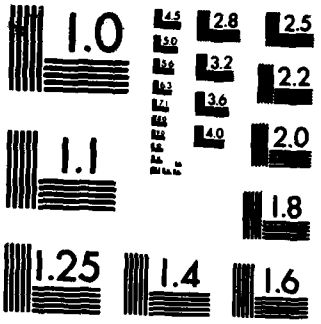
AD-A150 558 A DESIGN ANALYSIS AND IMPLEMENTATION OF A USER-FRIENDLY INTERFACE FOR THE UNIX OPERATING SYSTEM(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA F E GROENERT JUN 84 1/2

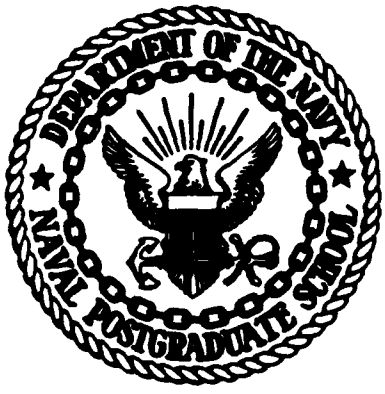
UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

A DESIGN ANALYSIS AND IMPLEMENTATION OF A USER-FRIENDLY INTERFACE FOR THE UNIX OPERATING SYSTEM

by

Frederick Earl Groenert, Jr.

June 1984

Thesis Advisor: George A. Rahe

Approved for public release; distribution unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A150558 / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Design Analysis and Implementation of a User-Friendly Interface for the Unix Operating System	5. TYPE OF REPORT & PERIOD COVERED Master's Thesis June 1984 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Frederick Earl Groenert, Jr.	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943	12. REPORT DATE June 1984 13. NUMBER OF PAGES 158
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

User-Friendly, Human computer inteface, man-machine communication, computer command language, man-machine interface design

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The user interface is a crucial, but often overlooked, part of the computer and software package. It is often the last thing designed. Short term memory aids and session pacing are two of the most important areas in which the machine can assist the user. Inclusion of a screen pointing device brings the computer closer to communicating on human terms. Basing the interface on a common metaphor, e.g. a desk top, can make it easy to learn and use.

(Continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT (Continued)

In this thesis three facets of the interface are examined: Communication from the machine, to the machine, and the dialog between user and machine. The Amiable interface designed for the UNIX operating system is described. Amiable is implemented on a SUN model 150 Workstation in the Naval Postgraduate School Computer Science Laboratory. Interface design is a hard problem, much remains to be done.

Approved for Public Release, Distribution Unlimited

The Design Analysis and Implementation of a User-Friendly Interface for the UNIX Operating System

by

Frederick Earl Grcenert, Jr.
Lieutenant Commander, United States Navy
B.S.E.E., University of Colorado, 1972

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1984

Author: Frederick Earl Grcenert, Jr.

Approved by: _____ Thesis Advisor

_____ Second Reader

_____ Chairman, Department of Computer Science

_____ Dean of Information and Policy Sciences

ABSTRACT

The user interface is a crucial, but often overlooked, part of the computer and software package. It is often the last thing designed. Short term memory aids and session pacing are two of the most important areas in which the machine can assist the user. Inclusion of a screen pointing device brings the computer closer to communicating on human terms. Basing the interface on a common metaphor, e.g. a desk top, can make it easy to learn and use.

In this thesis three facets of the interface are examined: Communication from the machine, to the machine, and the dialog between user and machine. The Amiable interface designed for the UNIX operating system is described. Amiable is implemented on a SUN model 150 Workstation in the Naval Postgraduate School Computer Science Laboratory. Interface design is a hard problem, much remains to be done.

TABLE OF CONTENTS

I. INTRODUCTION

A. THE INVISIBLE PIPE

The user to computer interface is the invisible pipe through which the man and machine conduct a dialog. Communication flows between the man and machine in the form of electronic signals, visual stimulation and possibly audible tones or patterns. The job of the interface is to translate the raw data forms into forms intelligible by the parties on either end. The greatest single benefit of a modern user interface is the ease with which someone may gainfully employ the system from his earlest session. The computer combines the role of instructor and tool; the best interfaces do it transparently. With appropriate aids the machine can anticipate the novice's misunderstanding, and offer explanations and guidance; yet accommodate the expert's impatience by streamlining his request. The ability to do these things is not an innate feature of exotic state-of-the-art hardware, though hardware makes them possible. The willingness of the machine to do the man's bidding is resident in the design of the man-machine interface and the software that makes it a reality.

Unfortunately the human aspects of users, even within a narrowly restricted group, are diverse and variable. What suits one individual is anathema to the next. Preferences

change periodically, and intellects grow and mature. The interface designer must accommodate the widest range of users in a manner appealing to all and condescending to none, a formidable task! The power of modern computers is beginning to allow the designers to bring the machine up to man's cognitive level, like never before. Techniques and equipment are being combined that offer vastly improved means of working, learning and playing. The laser video disk player (LVD) is a prime example. It is the answer to an interface designer's prayer. The LVD not only offers enormous storage, but can be interactively computer controlled to show selected combinations of instruction, explanation, and examples that would otherwise require a dedicated instructor or extensive reference material. It is the perfect medium to store the myriad error messages and explanations that previously were too numerous and cumbersome to attempt. With well designed lesson material, instruction at the workstation simply becomes a matter of selecting the disk, inserting it in the player, then hanging on for the ride. An LVD combined with a computer work station becomes a combination library/classrom/office with nearly unbounded potential.

The goal of the project with which this thesis is associated is to design a Command and Control workstation with off-the-shelf commercial hardware. This thesis explores the briarpatch of the human-computer interface pursuant to that end.

B. DESIGN FOR HUMAN-COMPUTER INTERACTION

It is not enough that people want to use a computer, their desire must be sustained by the human-computer interface before the machine will gain wide acceptance. The successful interface makes the user feel comfortable and relaxed while doing his job, and is faster and more accurate than conventional methods. A typical user will not change his or her normal work habits to use a new computer unless it is easy to learn and simple to use. Given a choice, users must perceive vast improvement over old methods before they will seriously consider the change. A little improvement is not worth the effort to change established methods.

The new generation of computers from APPLE, i.e. LISA and MACINTOSH, are exiting, relatively inexpensive examples of just what is possible in a modern interface [Ref. 1]. The dramatically new interface using icons and object oriented dialog appears to be attracting previously disinterested people to computers. Acceptance among traditional computer users has been slower, largely because the new interface is unconventional, and they are resistant to change.

Some display interfaces are easier to learn than others. Interface software complexity increases rapidly with user-friendliness. An interface that appears simple masks internal detail and complexity. The software running behind it is handling the minutiae formerly left to the user. But

the ease of use of some of the better systems will pay for itself many times over in reduced training costs and increased operational efficiency. It may be possible to save a few thousand dollars per system in the procurement process only to spend ten times that amount in recurring training costs.

In this thesis the three basic parts of human-computer interaction: user-to-computer communication; computer-to-user communication; and the dialog between the two are examined in some detail. An interface designed to help students new to UNIX is proposed. It is an overlay that works in conjunction with UNIX, not in place of it. Conclusions regarding the human-computer interface and future improvements are discussed last.

II. COMPUTER TO USER COMMUNICATION

The computer communicates to the user via output devices. A typical computer work station has one or more CRT (cathode ray tube) screens and access to a printer. Other output devices are possible, including voice synthesizers and other specialized hardware. A printer may be part of the installation, but printers are usually centrally located to be shared with others. Before CRTs became the accepted interface medium, printers were. Now they are used as copy machines: to obtain copies of intermediate and final results.

Audible bells or tones are used to sound fault alarms, or to warn of illegal operations. Frequently, the audio tone is felt to be an annoyance by the user community.

Voice input is an emerging technology, with high prices and low performance, but the cost to performance ratio is improving as research progresses. The difficulty of extracting meaning from spoken words and of joining sounds to form understandable words takes substantial machine resources and time. That additional processing slows the dialog between user and machine below a comfortable threshold. Computer vocabularies are small, ranging from a few words to two to three hundred words.

Voice recognition is used as a security check. Speech synthesis is also a major research area. Inexpensive speech

synthesis microchips have allowed the capability to be used in several electronic toys. The "voice" is still too mechanical and artificial sounding to be used in a general purpose interface. There is a tremendous future for voice input and output when it becomes cost effective.

A. THE PHYSICAL MEDIUM

The CRT screen is the communication interface. The information displayed there must be timely, accurate, meaningful, and concise. It is here that the computer must condense as much information as possible to keep the user informed about his current status, the status of his active jobs, and any other pertinent system information and messages.

Opinions about various monochrome screen phosphors are many and varied. Personal preference is the determining factor for which is best. All monochrome combinations (amber text, dark background; green text, dark background; white text, dark background; and reverse video combinations) are adequate, though the lined background of full screen reverse video may be less comfortable to use on text oriented raster scan displays. The most annoying distractors are reflected glare and misadjusted brightness and contrast. All these can usually be controlled by the informed user. Means should be incorporated in the terminal initialization sequence so the user can set his preferences.

Color monitor screens present additional problems. There are definite color combinations that produce remarkably uncomfortable visual sensations (green text on a bright orange background is one). Designers must be made aware of the vision function of accommodation and take pains not to inflict awkward combinations on unsuspecting users. Ideally, the user should be able to select his own preferred color combinations, then the designers problem is just that of providing the flexibility to do so. Default color combinations should be comfortable to view.

B. THE SCREEN DISPLAY FORMAT

There are two screen display formats used in general purpose interactive workstations, vector drawing and raster scan. Two versions of each are widely used.

A refreshed vector drawing display has high resolution but has been restricted to use primarily in graphics applications, where its special features are required. It is considerably more expensive than the same size raster display. The vector drawing display's forte is line drawings. It draws true circles and diagonal lines, because each line is a continuous stroke of the electron beam. The number of displayed segments is limited by the beam drawing speed, screen refresh rate, and amount of storage dedicated to vectors. The maximum number of displayable vectors is fixed, which is the primary shortcoming of refreshed vector drawing displays. Vector drawn display text is very sharp;

but because each letter is a combination of strokes the number of characters is limited. Consequently, text included on a screen frame rapidly depletes the number of vectors available to draw figures. Because of this only special purpose workstations have vector drawing CRT screens. CAD (computer aided design) workstations typically use a vector display for drafting and drawing functions.

The second version of vector drawn display, the DVST (direct view storage tube), eliminates the screen refresh. Hence it is much less expensive. DVSTs are not satisfactory general purpose workstation display surfaces because partial screen erasure is not possible, and writing speeds tend to be slow. DVSTs are excellent adjunct graphic displays for line drawings and text. Improvements and cost reductions in raster display technology have relegated DVSTs to a support role entirely.

The first kind of raster scan display is text oriented. The screen is nominally divided into 24 lines of 80 characters each. Screen area not contained in that pattern is not addressable. Each of the character positions is addressable and may contain any printable character. This screen is best used to display text. The main advantage of this display is cost; it is the least expensive to produce. A screen buffer need only be 2K bytes. It is possible to have two or three screen buffers to allow rapid screen filling operations, even at low data transfer rates. Low screen resolution, which results in poor quality graphics,

is this display's main disadvantage. Contrived bar and line graphing are possible, as is crude curve plotting.

The second kind of raster screen display is known as "bit-mapped" or "dot addressable". On this display the smallest addressable screen position is known as a pixel (PICture ELement). The physical size of a pixel is determined by the physical construction of the CRT and the signal handling speed of the amplifiers that control the electron beam. Very high quality, but expensive, bit-mapped color displays of up to 1024 x 1024 pixels are made. Monochrome displays can be had up to 2048 x 2048 pixels. Though more expensive than text oriented raster scan displays, the bit-mapped display is the display of choice for a computer terminal or work station. In the graphics mode it does suffer from "aliasing", the stair case appearance of diagonal lines and edges of circles. But software, and now hardware, smoothing techniques can minimize that effect for all but the most critical applications. The bit-mapped display does require a substantially larger screen or frame buffer (a Megabyte in the case of the 1024 x 1024 screen). But rapidly falling memory prices have helped to mitigate that disadvantage.

The raster scan format offers the most flexibility with the fewest restrictions in a general purpose workstation. The preferred raster display is bit-mapped. The workstation that uses graphic communication techniques should be equipped with a high resolution bit-mapped raster scan

display. It provides excellent character definition, good quality drawing features, and exemplary text legibility.

C. THE SCREEN LAYOUT

Most screens are arranged like a paper page. The beginning of the page is at the upper left corner. The screen fills from left to right, top to bottom. Only one page, or portion of a page is displayed. That standard layout has served well and will continue to do so. One display line is reserved for machine messages. Thus if the screen displays 24 lines, 23 may be used to display text.

The problem with the single sheet of paper per screen is that it does not offer a large enough "desktop", or working space. Few people work with just a single sheet of paper before them. Even the most rapid paging or scrolling screen is not adequate for reviewing material not included on the current screen. The root of the problem rests in the frailty of human short term memory. That shortcoming will arise frequently in the design considerations of a human computer interface. When the context of interest is lost because of changing the screen contents, all but a few basic facts about the previous screen are gone too. The solution to this problem lies in preserving some or all of the previous context when the user must change screens to see other information.

One answer has been to partition the CRT screen into several non-overlapping zones. Each zone is reserved for a

particular type of information. One might display the mode of operation, one could be a reserved area for error messages, another the title of the current file, etc. Fig. xxx illustrates a typical screen layout using zones. The zones remain on the screen, when the display changes, or the user changes what he is doing.

"Split-screen" is another version of screen zones. In this mode the CRT screen is divided into two or more distinct areas. Each area may contain something different. Two or more files may be displayed simultaneously. Two parts of the same file may be shown juxtaposed, to save the user from scrolling back and forth (and forgetting between). The problem with split screen operations is that the amount shown on each portion of the screen is reduced, often making it difficult to retain the context.

"Windowing" displays are currently fashionable. They are a trick way of getting more surface area from a fixed size screen. They are similar to the previously described zones; but are temporary, relocatable, and may overlap. They appear only when requested by the user, thereby saving screen space. Windows overlay and partially obscure the previous screen contents; but may be moved by the user to "uncover" an area. The currently active window is displayed on top of what appears to be a stack of pages on the screen. On most windowing systems individual window size and shape may be set by the user. Windows allow additional information to be available to view at the user's request,

while preserving the screen contents. The mechanism of the request can be any of several popular methods.

L. THE INTELLECTUAL LEVEL

The intellectual level of machine-to-user communication has historically been biased toward the machine. Inherent machine limitations and meager hardware resources dictated a low threshold. Early interactive systems responded with an "Unblinking Stare" to almost all user input. The only clue that his job or command had been received and understood was another "ready" prompt - a symbol appeared on the screen notifying the user that additional input could be accepted. Mistakes in the input often meant no response of any kind. Surprisingly, many popular systems still offer only that enigmatic stare, UNIX for one.

The intellectual level at the interface has risen rapidly with increasing microcomputer power and decreasing hardware costs. Fast, capacious storage and modern techniques have solved the former hardware deficiencies. Hardware is no longer the major limitation. The emphasis for improved performance was not directed to improving the interface until very recently. Whole CPUs (Central Processing Units) are dedicated to screen management and input-output; but the machine-to-user communication remains the same, with the exception of more accurate error and fault condition reporting.

Many of the faults with the interface can be traced to practices and procedures that began in the early days. Surprisingly, many batch processing procedures are common in interactive computing. For example: every computer programming language popular today was designed for a batch processing environment, being separately compiled, loaded and run.

The move away from the blank stare mode began soon after the advent of interactive computing. It did not receive much attention until the microcomputer explosion caused the computer to move from the laboratory into offices. Computers began to be called upon to do useful work in real-time. The clamor for user friendliness began. Many ideas have been tried and discarded. Those that have worked have appealed to the user's physiological, psychological, cognitive and intellectual needs.

The semantic level at the interface has moved from the bits to bytes (computer words, language characters), to command words, to clumped commands, and finally to a metaphoric level. [Ref. 2] The metaphor is a model that selectively combines all the useful and practical features of computer interface design and integrates them into a cohesive package that emulates a familiar environment. All available computer functions are masked behind a model of the application it is intended to serve. The user sees only terms, functions and methods with which he is familiar. The bewildering and intimidating world of the computer scientist

is completely hidden. The ideas that grew into the metaphoric interface originated with the pioneering work of Alan Kay and his group at XEROX PARC with SMALLTALK and Dynabook. Several commercial products are using these concepts, among them Xerox STAR, Dolphin, and Dorado; and APPLE's LISA and MACINTOSH [Ref. 1][Ref. 2]. This interface more nearly emulates a conventional desktop. Everything on the screen is treated as a physical entity, having an identity and occupying space. The keyboard is used only for text entry. A pointing device (mouse) is used to manipulate the objects. The translation is so complete and the implementation so natural that comparatively little effort is required to use the system. This object based display format is a paradigm of computer-to-user communication that will become a starting point for all new machines. The metaphor can be modified to meet the needs of many situations and applications, changing the nature of the computer from special purpose machine to general purpose tool. The keyboard is the final hurdle to widespread computer acceptance. Its elimination as the primary means of entering text and data will mark the beginning of an era that will see computers become as ubiquitous as televisions and telephones.

F. THE COMPUTER SPEAKS

Not every computer interface designed now will emulate the metaphoric paradigm. Designers must endeavor to keep

the level of computer-to-user communication as close to the human domain as possible. Every expert agrees that the machine must respond as quickly as possible to an input request. If it is unable to respond in less than one or two seconds, then it should notify the user that there will be a short delay, and give an approximate time to completion [Ref. 3]. If the processing will take a long time and partial results can be displayed, then the machine should begin responding as soon as it can with intermediate output. The time limit of one to two seconds is a rule of thumb, but fits well with human cognitive processes to maintain the user's train of thought [Ref. 4]. Every utterance made by the machine must be self-explanatory. Here utterance means "any computer-to-user communication". Cues prompting for input should, at the user's discretion, display the form (syntax) and options of the expected command. That helps the user two ways. First, it helps him remember the command, reinforcing his knowledge about it. Second, it reduces the probability of mistakes. Ideally the computer will display the command line, and the user can fill just the blanks, saving extra typing. As the user becomes more confident and experienced he can shift to a more independent mode.

A machine initiated mode of control relies on the form and substance of machine communication. The machine usually controls the dialog options in this mode; but it need not appear to be in control. The user is free to select from a list of options made available by the machine. The

restrictions placed on user action by this mode reduce the opportunity for error, both in number and type. Novice, inexperienced, rushed, and careless users all benefit from this feature. It also has disadvantages. It is usually slower, because the user must scan lists of, possibly unfamiliar, options and make selections. The selection mechanism adds to the overhead of the chosen operation. This mode is more cumbersome to use, because the control path is predetermined, and shortcuts are not easy, perhaps not possible. It will affect user flexibility, because the commands and options offered will tend to be general purpose. This mode is preferred for casual or inexperienced users primarily interested in accomplishing the task at hand. It can also be useful in system tutorials for new users.

Machines that do not have a state of the art user interface are not hopeless; but they do need help. A software interface package can be overlayed upon the operating system. It need not adversely affect system performance because it will be active only when the system is between modes. It will take space and slow parts of the present system. The idea is to display menus, soft buttons and pull-down windows on the screen. The display contents are cues to user choices. For instance, a computer user expects to be able to look through his files; write, edit, compile and run programs; compose and edit documents; copy programs and documents (electronically and printed); and get

on and off the system. Regardless of whether the system commands to do those things are easy to learn and remember, the new user is unlikely to know the correct command and therefore is unable to use the system. Yet he knows what he wants to do and how to do it manually. He is missing the key to unlock the power of the machine. The claim is: He should not need specific knowledge about the machine, or the operating system to use it. The user is normally not interested in processing details, only in getting his job done. He should not be burdened with computer-specific details. The machine is supposed to make his job easier. In fact, present computers add a layer of complexity to most jobs.

The interface overlay easiest to implement is menu driven. It can be enhanced to a full fledged metaphoric interface if the machine is sufficiently fast and has the resources. A pick device (mouse) is preferred; but not essential. The pick decreases reliance on the keyboard, decreasing user apprehension. All menu selections are labeled with generic terms, recognizable by a broad range of users. The specific system is hidden by the functional mask of the interface, directly embodying the first Parnas' principle of information hiding [Ref. 5]. The overlay need not, in fact must not, be static. A static interface would necessarily be slanted toward a specific user group. To increase the user domain, flexibility is essential. Instructional messages intended for a novice cause annoyance

and impatience in experts. The same flexibility should also allow the expert to call on available aids, if he so decides. Though the machine controls the number and type of commands allowed the user is still in command.

F. ERRORS

Error handling and reporting is the most crucial facet of man-to-machine communication. It is easy to inspire user confidence and calm when everything is working correctly. The test of an interface is how gracefully it behaves when something goes wrong. The classes of errors are defined for the typical computer system. Some more exotic error treatments will be touched on afterwards. Three classes of errors are readily definable: superficial, recoverable, and fatal.

1. Error Types and Remedies

Superficial errors are caused by user carelessness or confusion. They include typographical mistakes, mistaken selections, and out of sequence actions. They are easily detectable and correctable, in most cases. Typographical errors may be caught by either the user or the machine. Mistaken selections are legal operations, but not the one the user intended. The onus is on the user to correct the situation, but the system can help by allowing the user to quit an operation or sequence at any intermediate point. Out of sequence actions occur when a user fails to do a necessary intermediate step. For instance, he tries to

execute a source file without first compiling it. This error is preventable, but the designers must include explicit features to do so. They must try to anticipate user actions, providing appropriate reminders and procedures that illustrate the proper sequence. If the user learns by his mistakes he is less likely to commit the same error twice.

Recoverable errors are not discovered until after the fact, but may be undone by some means to get back to the starting condition. This is a much larger class of problems and requires more ambitious methods to fix. The methods usually involve restoring words, lines, or small files that were erroneously changed or deleted. The recovery techniques involve intermediate back-up. Small entities like words and short files can be temporarily saved in a buffer until the proper result is confirmed. The storage overhead is not unduly expensive, relative to the confidence the ability to undo the operation instills in the user. Larger files can be backed up on disk or tape, but the user must usually take overt action to do so. A small class of functions may be inverted, i.e. run backwards, to retrieve the input from the results, but that is an exception.

The final class, fatal errors, are non-recoverable, or are so difficult to recover from that it is not worth the expense. These errors must either be prevented, or the user warned of the consequences before they are executed. For example: a user decides to delete the entire contents of a

directory, and imprudently uses the command "delete *.*" without realizing he is not in the directory he intended to delete. The command is legal and the system will execute it. The system designer must have anticipated this situation and included a measure of protection. To warn the user the system should respond with a message similar to:

> "WARNING, delete *.* will delete all files in directory:
>
> <directory name>
>
> Confirm intention by pressing the left mouse button. Press the right mouse button to abandon command."

Where <directory name> is the current directory. This gives the user an opportunity to ponder his action and reinforce or abort his decision (and save himself considerable anxiety).

Another type of fatal error is caused by hardware failures. These errors are becoming less common, but can be heart-stoppers when they do occur. The only remedy for this class is a comprehensive back-up program, expert technical support and a dose of good fortune.

2. Ideal Error Response

The ideal response, in most cases, is no overt response to the error condition at all. The machine knows what the user intended (based on the context in which it occurred and because the suspect word nearly matches, or is an alias for, a known word) and carries out his request, ignoring the user's misspelling or command synonym. A modern

interface design should specify a more partitioned response. A new user, or one learning the system should be presented a reminder of the proper spelling or command. That could be done by having the system substitute the correct command word at the appropriate spot, for instance. The Interlisp system perceives the user's intention, requesting confirmation of spelling corrections it makes automatically [Ref. 6]. In operation it seems uncanny, as though another person is watching over the user's shoulder. Interlisp demonstrates the power available within the machine, but the resource costs associated with it make it impractical for most microcomputer based systems. A more accomplished user might turn off the reminder feature if it annoyed him, or if it sped up his job. An even better method is not to use the keyboard at all except for entering text in a document or names into a program. Compiler errors will become anachronisms, as more syntax directed editors become available. The emphasis is now turning to run-time error detection and correction, a much more troublesome and difficult problem.

To be helpful the machine must provide enough information about the error type and location so that the user can attempt to find and correct it. That implies the computer must have knowledge of the program, as well as raw information. Unfortunately, the artificial intelligence program required to perceive the user's intention from all the data on hand would seriously overload all but the most

powerful microcomputers. The machine usually has sufficient raw data about the error to provide accurate information about the problem. The information is usually not in an understandable form. System designers have been reluctant to provide the facility to do the jot of translation, because it has been considered to be wasteful of precious resources. For example, many modern error handlers respond to an incorrect address call with: "Bus error(core dumped)". which sends the user back to his program; or worse, to a hexidecimal core dump, to find the fault. Most systems are capable of: "Address out of bounds in subroutine <subroutine name> statement <statement number>", if the error handler has had the proper emhasis from the beginning of the design. An artificial intelligence based debugger might respond as in the previous statement with the addition of: "... expected <address name>". The ultimate response might be a warning to notify the user to verify the corrected address. In the recent past the designer's feeling has been: "A minimal error handler is adequate because real programmers will figure it out." That is no longer an acceptable attitude on any design team. Those resources are now expendable, translating data from one form to anotner is one of the machines strongest attributes.

On line debugging tools have been some of the most obtuse program development aids ever devised. Single source-statement stepping techniques (from compiled code) are some of the easiest tools to use, but are not widely

available. That is the normal debug mode of interpreted code.

3. A Specious Notion?

The notion that mistakes can be reduced by providing rewards for correct performance and by punishing incorrect performance seems to be apropos in learning to use computers. The concept of rewards and punishment is well founded and widely accepted, but in this context must be extremely subtle. Initial attempts will probably fail. One proposal is to reward correct behavior and provide correct usage prompts when incorrect entries are attempted [Ref. 7]. In theory it sounds plausible. The particular implementation proposed by Dwyer appears to be too obvious. Most human operators will chafe under the notion that the machine is dispensing symbolic pats on the head. The issue is who is in control, the user or the machine? A better method is to allow correct performance to be rewarded with obvious progress. Incorrect performance should be difficult or impossible. Allow the user to set experience level keys or let the system recognize the user and set an experience variable internally. User selectable options are best. Various levels of experience invoke different guidance for each user action. The notion of rewards must be very cautiously implemented, and only after extensive testing.

III. MAN-TO-MACHINE COMMUNICATION

Man-to-machine communication has been the dividing factor between computer users and non-users. Past (and present) computer command and programming languages are cryptic, confusing, difficult to learn and nearly impossible to remember. People are not mentally equipped to learn things like that. It is especially frustrating to learn a particular set of commands and their idiosyncrasies only to forget them almost entirely during a fortnight of non-use. The reason for the inhuman lexicon in the past was lack of storage space. There was not enough space to do all the things that had to be done, so the sacrifices had to be made by users. The former extravagance of more understandable commands has been made affordable by the microcomputer revolution and the shrinking microchip. Sufficient memory and processor speed exist in all but the smallest machines. Note that the command languages transcend the mere typed code word or phrase. New methods of impressing the user's intention on the machine involve some very human abstractions, e.g. pointing to the item of interest. Regardless of the method involved the newer communication forms have been kept from wide use by the expense of hardware. That is no longer the case.

A. INPUT DEVICES

There are many kinds of physical input devices: keyboard, joystick, light-pen, mouse, digitizing pad, and touch panel, to name several. A more complete description is included in [Ref. 4][Ref. 8]. A brief description will suffice here.

The typewriter keyboard is identified with the computer because, even before widespread interactive computing on teletype machines, key-punch machines were used to create cards that the machine could read. It is the classical computer input device. As such it is a first choice by computer system designers. Most computer users expect a keyboard as part of the machine. It may have from 40 to over 100 keys. The upper and lower case alphabet and digits form 0 to 9 are included, as are most punctuation marks. Most keyboards are encoded with the ASCII (American Standard Code for Information Interchange) character sets. The other standard character set is EBCDIC (Extended Binary Coded Decimal Interchange Code), used primarily by IBM. The ergonomics of keyboard design was decided before the advent of the computer. It is a result of typewriter design, and is locked into its familiar, but awkward QWERTY layout because of the costs and politics involved in having people learn a new keyboard layout. Other designs, more suitable to human physiology, have not shown sufficient improvement over the standard to be considered for wide application.

The most likely fate of the keyboard is that QWERTY will be superceded by voice input.

The joystick is a holdover device from the era of analog computers. It is a stick mounted with two degrees of freedom. Motion in either of two perpendicular directions causes a wiper arm on a potentiometer to move, changing the resistance. That resistance is converted into a digital signal and fed to the computer. Thus a joystick can represent a point in X-Y space easily,and so can be used to position the cursor. As a cursor controller the joystick is a member of the locator family of logical devices.

A light-pen is a light sensing device used to denote a point on the screen. It senses the light emitted when the electron beam strikes the screen phosphor. It is shaped like a pen and has a thin wire bundle attached to the end opposite the sensor. It shines a light from the sensor end to show the user precisely where it is aimed. A light pen is most often used as a pick, though it may also be used as a stroke, locator or valuator device. The light pen has fallen out of favor because it is more expensive and fragile than a mouse. Because it is sensitive to all light, not just the light generated by the CRT, it must be kept in a holder or stand when not in use. That means the operator can not just set it down between uses, like he can a mouse or joystick. Another light-pen failing is it may become a missile hazard if used in a non-benign environment, e.g. a ship, aircraft or automobile. The light-pen was once a

breakthrough in computer control. It is being replaced now by the mouse, a cheaper more robust device.

The mouse is the new darling of the computer interface designers. It offers a flexible and relatively inexpensive, means to control a screen cursor. The cursor is then used for most interface functions. The mouse is more expensive than cursor control via a keyboard; but it enables a control mode that is much more closely attuned to human cognitive processing [Ref. 4]. Consequently, it is much easier to use. It also suffers the missile hazard problem of the light-pen. But a user may leave the mouse where he last used it, without fear of generating spurious input, as the light-pen might. The number of buttons to place on the mouse is arguable. Three buttons seem to be widely accepted, but one and two are also used. The decision should be made on the basis of the expected user group. The arguments range from "One button may be less confusing" to "The software may be easier to write for three buttons." [Ref. 1]

A digitizing pad is used to translate graphical information into a machine-readable format. It is a flat surface with either a mouse or stylus connected. Drawings or other graphics information are placed on the surface, then points are read from the drawing using the mouse or stylus, digitally encoded and transmitted to the computer. A digitizing pad can double as a mouse type cursor controller.

Touch panels are a recent addition to the world of computer input devices. Two different technologies are used. One uses transparent pressure sensitive membranes adhering to the surface of a display screen. The other version is made of two linear arrays of laser diodes, arranged at right angles, that form an invisible grid of intersecting light beams on the display screen. A finger, or any other opaque object that interrupts one or more beams is interpreted as an input signal.

The major disadvantage of touch screen panels is resolution. To keep costs within reason, and to offer acceptable resolution (approximately three or four lines per inch) the touch panel is suitable for use as a pick device. Higher resolution is possible, but costs escalate rapidly. If resolution is finer than a human finger tip, a stylus must be used to indicate user selections, to avoid ambiguity and spurious inputs.

The ultimate in cursor control is a "no-hands" controller. Implementation is by means of a hat containing a sensor that determines where the user is looking. Most use a low power laser reflected off the eyeball to read eye motion, which is translated into a screen position. This mode of operation allows the user to keep his hands on the keyboard, a benefit for accomplished typists. No-hands cursor controllers are just now becoming commercially available.

B. COMMAND LANGUAGE

A command language is the means by which the computer is made to do useful work. A typical computer command language is designed to implement functions. Those functions may be simple or complex, low level or high. Language design based on grammatical, syntactic and semantic properties of human language tends to generate computer command languages that look like spoken language. In the past hardware limitations drove command language design decisions. Now, new hardware capabilities are allowing a totally different language model in man's quest for more natural computer use. The command language abstraction is moving rapidly toward the human end of the man-machine communication channel. The trend is away from command words and phrases, and toward a command representation that emulates the environment. The things the machine can do are being expressed in common, understandable terms. In some, words are not used at all, pictures and motion, based on the well known metaphor of a desk top, suffice to define an unambiguous action the user wants the computer to execute.

Command languages may take many forms and be implemented several different ways. Machine control methods below the process logical level will not be discussed here, because those details are not useful to the general user. In this context a process may be thought of as the logical embodyment of a user command. Typical processes are: logging onto the machine, editing a file, editing commands within a file,

browsing a file, executing a program, etc. The most common command language implementations are enumerated below.

1. Simple Command Language

The simplest command languages use single character codes to represent legal operations. The best attempt to apply mnemonic tags; but that is difficult with single characters, e.g. E could mean edit or erase - two entirely different operations. Single character codes are no longer used as computer command languages; but are still widely used in interactive editors, debuggers, and various utility programs. Single character codes minimize typing.

The next most simple command structure uses multiple character command names, usually between two and five. Mnemonics can begin to help users differentiate commands; but they still need handy reference material.

Full word command languages are the easiest to remember. The functions are invoked simply by entering the name just as before; but now the function name is fully specified. Spelling and abbreviation peculiarities are no longer a concern in remembering a function name. This mode can be enhanced to save typing. The machine can fill the trailing part of the command word or phrase as soon as it recognizes an unambiguous leading character string.

The factors that separate a good command language in this category, from a bad one rest in the syntax and semantics of each. Syntax rules like the "zero, one, infinity principle" [Ref. 5] , and command structure

consistency can be important aids to learning and use. Ignoring simple rules like that can result in an abomination. Learning to use simple command languages is almost entirely a memory function. The user must associate the thing he wants to do with the correct function name and syntax. The early stages of the process are very frustrating because the user must constantly refer to documentation for help. Even the most regular languages still require a learning period.

2. Function Keys and Programmable Keys

Function keys are the hardware designers answer to the difficulties of learning command languages. They are usually additional keys on the keyboard, that may or may not be labeled. Function keys relieve some of the short term memory problem but they add to the user's initial confusion and intimidation. Some the keyboards in systems that use function keys are awesome. After learning the extra button meanings and locations they are generally easy to use. They also tend to limit system flexibility, because the user has no control over the functions. The inflexibility is felt in other ways too. Users who initially use the system because the function keys are self explanatory often find themselves disenchanted when they have outgrown the novice-user stage.

Programmable function keys are a partial solution to the problems of function keys. They allow additional flexibility for both the system designers and in most cases the users. The flexibility comes because each key is no

longer locked into a single function. Each application program may redefine the keys to suit its needs. In many cases the user can define keys to suit his needs. Fewer function keys are needed which reduces the users initial awe. Unfortunately, now the user is faced with a memorization problem again, unless the designers provided for variable labels too.

Function keys and programmable keys are an attempt to compact more functionality into a fixed capacity. They are a compromise, they add to equipment costs, and they make the system unique because of the added keys. Someone used to a general purpose keyboard may have difficulty adapting. That is not a failing of the hardware, but a user interface consideration. Systems with fixed function keys tend to be aimed at novice computer users. That translates into: Only novices will use the system. Too much attention paid to a particular user segment usually means the rest get neglected.

3. Menus

Menus are the software designers answer to the short term memory problem. They have advantages over the function key solution, but they also suffer some of the same flexibility problems. They can be variable size and length. They are more self-explanatory than keys, because the designer can write more than a single word or abbreviation in an entry. Like keys they also channel the users activity into predefined paths that may be difficult or impossible to

depart. Designers must be careful not to take the menu as a final solution to command language problems. They are a useful tool and when used in conjunction with other devices can help create a powerful environment. They can also be a nuisance. Escape sequences from several layers of menus can be particularly devious if the user is not kept informed of his whereabouts in the hierarchy.

4. Macro Instructions

"Macro instruction" is a term coined to define a combination of computer commands, used frequently, that may conceal intermediate results. The term was first applied to assembly language programming; but is used at all levels of computer language now. Macro instructions are the key to user flexibility. They are an easy way for the user to customize a system to his personal desires. Unfortunately they are some of the hardest things to learn about a system. The reason is somewhat self perpetuating. New users rarely write their own macro instructions, though they may use some give to them without realizing it. They usually are unaware of the existence of macros. Experienced users use macros extensively. They also write the documentation about how to write and use macros. Have you ever tried to read system documentation? If you have, now you know why only experienced users use macros.

5. Icons

Icons transcend written language. In the context of computers they are small uncomplicated sketches that

represent commonly known ideas. Understandable ones are symbols that represent a particular metaphor. Icons are all around us in life. They are especially helpful when trying to cross a language barrier, as is demonstrated by international road signs. An unfamiliar user may not be able to decipher an icon at first; but once he knows its definition he will remember it when he sees it again. They are the easiest interface device to use; but place heavy demands on hardware resources. Icons require bit-mapped screen and fast secondary storage, as well as plentiful primary storage.

6. The State of the Art

As stated, the departure from conventional computer command languages is a result of vastly more powerful machines, especially microcomputers. Object-oriented interfaces have captured the imagination of the industry. They embody much of the novelty and "gee whiz" excitement of arcade video games, yet are aimed at entirely practical purposes. "Wizzywig" is a term for this family of command languages, from the phrase: "What You See Is What You Get" (WYSIWYG). The user is not required to bring anything to the machine but common sense and a job he wants to do (which the machine is capable of). The language he uses simulates manually doing the task. For instance, to produce a new document the user selects the picture of a typewriter from the icons displayed. Upon selection of that icon the computer responds by entering the word processing mode, at

the top of a blank page. The user may begin typing as soon as the displayed page is in place. The keyboard is his typewriter, the screen his paper. Character and word delete keys are used for typing corrections.

Larger editing functions, e.g. line spacing, font styles, text insertions and deletions, are mouse controlled. The function is selected with the mouse. Then, begin and end text positions marking the range of application are chosen, also using the mouse. As the mouse button is clicked, confirming the range, the function is executed. The results appear in the text, without any further user action.

The wizzywig editor described here is the state-of-the-art in word processing software. It is not inexpensive, but is affordable. The APPLE MACINTOSH is sold with a version that, while restricted to six inch lines and ten pages, offers all the capability described above. Note though, the research and design effort that developed that editor was the culmination of over a decade of active pursuit, and the editor is not transferrable to other microcomputers outside the APPLE family of "32 bit machines". It does demonstrate what is possible in a fairly modest but capable microcomputer-based system. Sadly, this class of editors are not generally available because they do not exist for most machines, yet.

IV. The DIALOG

User confidence in a computer system depends totally on how he views the system. That confidence builds in stages beginning with the user's first experience with the machine. If the hardware features are close to his expectations his confidence grows; hardware upgrades can be had later. If the software is available for that machine to do what he wants and needs, he is again bolstered; software is relatively inexpensive. If the user interface is awkward the machine will gather dust. Some people are willing to make allowances for every part of the machine except the interface, the part they can not touch or really see. People, especially those outside computer science, will not wrestle with a machine, even is it is supposed to be better.

If the user interface is crucial to machine acceptance, it is even more so for acceptance of software application packages. The feature of the interface that people perceive is the dialog, defined as the conversational mode of communication between the user and computer. Regardless of all other hardware and software considerations, if the dialog is clumsy, unevenly paced, too tightly machine controlled, or otherwise unappealing to the intended users they will perceive the machine to be "bad". They may not even be aware of why they feel that way.

Interface dialog must accommodate user psychology. The statement is simple, but the application of that statement is not. With skill and artifice the computer can be programmed to appeal to the human psyche, and that is what modern interface design is about ... MAGIC!

A. STARTING AT THE BEGINNING

Two distinct dialogs exist at the communication interface. Each is better suited to different situations, though neither is totally wrong in any case. The telling factors lie with both the user, his computer skills, and his intent; and also with the machine, its capabilities, and purpose. One dialog is computer initiated, the other is user initiated.

The machine initiated mode, mentioned in chapter II, is the predominant one in a modern interface. The strong attributes of the machine are brought to the interface to help compensate for the user's weakness. By displaying lists, the computer's accurate recall is used to nudge the user's memory about the environment, commands, logical position, and various other details the user needs, but has difficulty keeping under control. If the amount of assistance rendered is user selectable, the designer has placed the user in de facto control of his environment. If some of the levels of help happen to be tutorial, the system begins to emerge as much more than just a smart typewriter.

The second mode is just opposite, the user initiates the action via a command sequence and the machine responds. In its pure form, this mode relies entirely on what the user knows about the system. It is the most common man-machine interface dialog. The reasons are historical and economic. It requires fewer machine resources and, according to Foley and Van Dam [Ref. 8, p. 227], it costs less, because the machine need not store any information explaining commands in human terms. The burden is placed on the man to conform to the machine operating environment, by learning and remembering machine commands. Without the proper commands and rules to use them, the system is less responsive than a portable Smith-Corona electric typewriter hooked to a television. Modern interface design practices dictate limited use of this mode.

This mode does have its place in a computer interface. There are experienced users that will always rely on their own skills. The bare system will probably be faster, though mistakes will be more costly, in terms of lost time and wasted effort. Less experienced users may experiment with it, and ought to have the opportunity to access it when they are ready. Everyone should experience the thrill of working without a net.

Computer users need not despair until the wizzywig concept is expanded to generic software packages. There are many things that can be done, with some effort and appropriate hardware, to ease the pain of learning command

languages, or to conceal them from the user. The first, and predominant, hurdle to overcome is short term memory. The following memory aids can be implemented on any personal computer; hardware required is minimal. The degree of memory assistance to implement is a major design decision, for two reasons. In the early stages of learning a computer system, short term memory works at odds with the user's attempts to learn. He is constantly frustrated by his inability to remember commands, syntax, and minor details of system use. Memory aids at this learning stage may be either extensive and detailed, to give the learning user all the information he needs; or better, the system can hide the detail, keeping functions abstract and general, to relieve the new user from details he does not need or care about. The major design difficulties arise because the new user does not remain a novice forever. Within days or weeks his needs and wants will be dramatically different. Without care, memory aids can become voracious resource consumers.

1. Simple Memory Aids

The simplest memory aid is a displayed list of commands. The list removes the burden of keeping several barely familiar command names separate. Every possible command need not be displayed. A complete subset (one general purpose command for each available operation) is adequate. The displayed list is only that; the user must still choose the proper command, and enter it via the keyboard. Nothing more than the command word is shown on

the list. The only processing involved to support this method is writing the list to the screen and maintaining it there. This method is less useful to the novice, but is the least expensive to implement.

a. With Syntax

The first logical extension of the list above is to display representative command syntax. If several options are associated with the command, display the most general one. For example, the UNIX VI screen editor, 'delete' command might be displayed as: 'ndW'. The user can now begin to see some of the options that he may not have remembered from his reading. A casual user, who has not used the system for a few weeks, seeing the syntax example with the command is more likely to recall the syntax rules for the command (assuming the syntax rules are regular; they are in this case). The failing of these first two methods is that the user must know the commands before he gets much benefit from either form. Mnemonic command names can help; but something more is needed.

b. With Explanations

The next possible enhancement is to add a one word or short phrase of explanation for the command. e.g. 'ndW' - Delete n words. Two factors affecting the design must be considered: space and interpretation. This form takes more space. Fewer commands may now be displayed and still preserve adequate workspace. Careful decisions must be made to include enough information in as little space as

possible. Once again restrictions are imposed by the equipment, now caused by physical display area constraints.

Taken literally, a novice might misconstrue the command to mean it deletes words that begin with the letter 'n'. The designers must make some assumptions about user entry level reasoning and experience. That does not mean the design and implementation will be universally understood as intended. In this mode the class of errors generated by misunderstood cues is still a problem. How should the machine react to a literal 'ndW' command? A usage note displayed with a page of commands explaining that 'n' is a variable that may be omitted or replaced with an integer, may be sufficient to prevent the error. A beep or screen blink will probably not be helpful; because the user was explicitly following the guidance. Testing, using representative subjects, will show which aids are needed and where, identify common errors, and indicate possible remedies.

2. The Result So Far

What has been accomplished, and what has it cost? That depends entirely on one's point of view. Assuming the memory aids can be implemented inexpensively, they will help new and casual users to be more comfortable using or learning the system. (Inexpensive is a relative term, measuring against time and productivity lost while learning and relearning, memory aids are inexpensive.) Experienced users will probably ignore the aids, preferring to use the

area to expand the working context. The question of whether to display a memory aid by default or to leave it to user discretion lies in user-population composition. If the users are predominately clumped at one end of the spectrum, make the default selection suit the main user group; but make the facility selectable. The flexibility is important because it reinforces the notion that the user is in control.

The static memory aids discussed above are possible on most microcomputer systems. They are employed extensively by several commercial software products. The difficulty involved in implementation is screen management, which may involve some complex programming (the menu and work area must be displayed), but is worth the effort. Memory aids do not cause any fundamental command language changes by themselves. The user merely scans the list, chooses a command, and enters it on the command line. The method does not subvert any capability the basic system may provide, such as combining several commands together to suppress intermediate steps. A novice need not be aware of the features that allow shortcuts, but more knowledgeable users are free to take advantage of them. Man's burden has been eased in the early learning stages, but he learns low level system details as he gains familiarity. The first design implications of the span of differing expertise levels are beginning to emerge.

3. Suppressing Detail

The next level of abstraction employs information hiding and selection. Unnecessary system details are concealed from the new user. He is presented a list of operations expressed in generic terms. Command choices are made by typing the selection number or letter and supplying arguments - filenames, directory names, modifiers, etc. This is a more natural way to invoke high level operations like editors, compilers or document formatting programs. The user does not type the command name, only the selection designator. Larger operations invoked with less user effort separate this category from earlier methods. A single, user supplied, keystroke causes the computer to change operating modes. The system hides all detail involved in stopping the previous display, tidying up the loose ends, and starting the new program. The user sees the screen clear and the new screen appear, just as he expected.

A full screen editor is normally the first exposure a user has to cursor control. Ordinarily input is directed onto a command line, the top or bottom screen line. In a screen editor the cursor - a symbol (often an underscore, box, or reverse video rectangle) - may be moved to any legal character location. Cursor control is usually by direction keys, marked with arrows, or function keys. Each keystroke moves the cursor one position, left, right, up, or down. Provision is made for continuous motion if the button is held down. Any screen input is echoed at the cursor

position and the cursor moves to the next position. A cursor is nothing more than a symbol to focus the user's attention. The cursor can be thought of as a portable peephole into the machine. It shows the user where to look. If it is not where he wants to look, then he must move it. The only way the user can access the machine is through the cursor.

The features discussed up to this point are routinely found in commercial software products. A typical installation includes a microprocessor, 64K bytes of RAM (read-write, Random Access Memory), and one or two secondary storage devices, usually floppy disks. The next improvement requires a hardware upgrade, incorporating a "pointing" feature to give the interface a more natural feel for non-typists. These devices allow the user to symbolically point to something depicted on the screen. The user is able to select any screen location with a single coordinated eye-hand motion. Two hardware devices are well suited to this task, the mouse and light-pen.

E. A QUANTUM HOP

The pointing feature entails more than just buying a mouse. What appears to be a small enhancement means major changes to both hardware and supporting software to make up for the hidden details. Changing cursor control modes from single-step keyboard driven, to continuous pointer driven, means much more complex screen management. One can not

expect to upgrade a personal home computer to incorporate a mouse controlled display with windows and still retain acceptable response. To be sure the computer could do it; but winter molasses would move like the Concorde SST in comparison. To fully explcit the power of a pointer and concomitant windowing techniques requires a major upgrade in computer capability. Fluid display switching demands data rates well beyond that of smaller machines, as well as special display management hardware. The full system upgrade is substantial. The first commercial systems that offered this capability cost approximately $100,000 per station, in 1978. Prices have fallen dramatically, in 1984, an APPLE MACINTOSH sells for $2500, though much less powerful, it has a totally wizzywig interface.

1. Basic Mouse

As a pointer, pick, or locator, the notion of the cursor changes slightly. It is now a shuttered peephole, opened by some user action, for instance, pressing a button. When the cursor is shuttered it may be moved freely around the screen but has no effect, other than to attract the user's attention. This mode is similar to the cursor in the screen editor, but now all cursor control is from the mouse or light pen. When used as a pick the shutter is opened briefly and information is transferred from man-to-machine, and vice versa. The information transferred might have been a menu selection and response. The raw cursor position information was transferred in. The machine translated the

cursor position to the menu selection and executed that choice. As a locator, pressing the mouse button means: mark this point. The machine knows the implied operation because of the context in which the mark is given.

One might ask why one should go to the trouble to have this mode, since key control is fine and much less expensive. The mouse gives an intangible boost to the user because he can now accurately position his link to the machine where he wants it just about as fast as he can find it visually. Light pens and mice are Fitt's law devices. A user can position the cursor using them just as fast as he can point to the new spot, because the cognitive processing is nearly the same. Testing has shown the mouse and light pen to be superior to any other locating devices [Ref. 4]. As interface control moves away from the keyboard, the mouse or some other Fitt's law device will emerge predominant, a "no hands" controller might be even better.

There is a trap here for the unwary designer. If possible, allow both control methods of input. Use two cursors if necessary. Let users decide which they prefer. Experienced typists feel more comfortable with the keyboard and eschew the mouse.

What interface problems are easier now? The structure of the program that runs the interface is very simple now, but the program is long, and large (over 200K bytes of object code for one screen is not unreasonable). The user is presented with lists exclusively. He may only

choose one command at a time. The phenomenon of psychological closure is being accommodated [Ref. 3]. The user sees definite action-reaction pairs, choices have explicit results. The environment presented is most likely hierarchical, so while working the user is traversing a tree or network. The user does not need specific system knowledge (if the menu designers have done a good job). The user is better able to concentrate on the job at hand and not the machine.

There are still some shortcomings. Context is difficult to maintain from one menu to the next because the full screen changes with each new menu. Screen management is a tacit design challenge.

2. Windows

Windows are the latest addition to the interface designer's bag of tricks. They are a logical extension of the concepts of screen splitting, and zones, but are dynamic. A window is a generalized screen or pseudo display. Anything that can appear on a normal display can appear in a window. Windows can appear simultaneously with other windows. Good window performance calls for hardware and software support. The operating system must support multitasking, i.e. a single user must be able to run several active processes at a time. Some form of raster operation is necessary, as are fast screen buffer(s).

A raster operation is the ability to extract all or part of the screen buffer into a file and recall it to the

screen at will. Raster operations are very fast, the whole screen may be changed in one refresh cycle. They eliminate waiting for screens to fill with text of graphics, if properly implemented.

Two window styles are used, static and pop-up (or pull-down). Static windows are work areas, usually large, and contain text or graphics. Pop-up windows are used for displaying transient lists, primarily menus, and are small. A pop-up menu is only displayed while the user makes a choice. As soon as the choice is made the menu disappears and the choice is executed. That implies that more than one window may be displayed at once, allowing the user to preserve context as he chooses, like flipping pages. Both static and pop-up windows can be moved around the screen at the user's discretion.

C. GOALS FOR INTERACTIVE DIALOGS

The user perceives the system as the dialog between the himself and computer. The designer must make that dialog as straightforward as possible, in human terms. The following guidelines are a starting point for the interface design and implementation.

1. Minimize Apprehension

Attempt to minimize user apprehension and intimidation. Some uneasyness is inevitable on any machine, at first. Try to get the user involved in the work, through the dialog, making the machine transparent. Keep the

interface businesslike. Do not try to be cute, though dry humor may have a place [Ref. 4]. Keep error messages complete, but terse. Instead of evaluating user performance in messages, use descriptive terms to point out the problem [Ref. 3]. Avoid judgemental terms in messages: Instead of 'Illegal Command' which places blame on the user and makes him defensive, use a more descriptive phrase, like 'usage <filename> <mode> <options>'. A usage message shows the user the correct way to invoke the command. It also lets him see his mistake and easily see how to correct it. A menu driven system prevents the majority of errors like these, most of which result from careless typing.

2. Don't Panic

Minimize user frustration and panic. An uneasy user is more likely to make mistakes, especially if the same nagging fault is causing his displeasure.

(1) Panic results when the system does something unexpected then locks up, without telling the user what happened. Always keep the user informed about the state of the machine or process. A simple "running" or "stopped - address fault <process name>" is adequate. It should always be displayed at the same screen location or zone. If something does fail report it to the user, then lock up or autoboot.

(2) The user should never feel lost. Keep him informed about his location, either with a "world view" diagram or highlighted menu choices, etc.

(3) Always let the user know what options are available to him. Guessing can lead to disaster. Pop-up menus are excellent for the task. Otherwise display a menu unobtrusively at a margin or border.

(4) Accommodate several levels of expertise. Try to put the user in control here. No one likes to be classified, especially by a machine. A toggle for experienced users is one possibility.

(5) Make the user feel he is making progress. Stress psychological closure in the design and implementation.

Following the points above will make for more pleasant error situations, and if the user is more relaxed he is also more likely to recover with less trauma to his ego, and his work.

3. Natural Dialog

The dialog must be couched in terms familiar to the user. Whether it is words or pictures, they should be from the user's sphere of reference.

4. Solicit User Comments

Systems grow or die. User comments will tell the designers what is disliked about the system. Things that are good in the user's eyes are invisible. Annoying faults, no matter how small will not go unreported. Make sure the comments are read and acted upon by some one with the power to make a difference. Not all problems will be repairable, but at least they will be known. Feedback is especially important early in the design stages. Try to get actual users on a new system as early as possible.

5. Keyboard?

For applications that are not heavily text and data entry, or word processing try to minimize keyboard use.

Database manipulation applications are a likely candidate for a wizzywig interface.

6. First Session

If the system can not be used for useful work from the first session, something is wrong. A general purpose set of commands and applications should be available without any system knowledge other than how to login. A user who is able to sit down with no prior knowledge and leave a short time later with a completed document, and a smile on his face, will return. Explicit training done on the system should be done in short lessons separated by distinct breaks. Ideally the instructions should be designed for ten to fifteen minutes, with thirty to forty minutes for hands-on practice. Keep the user engrossed in the job at hand. If he must think about how to do something on the machine, he is not thinking about his job. The machine is just there to help, not dominate the task.

All the experts agree on what to do and what not to do. It is also obvious who among them have designed an interface, those who have tried, and those who have not. They agree that interface design is more art than science. It will remain so until computers become so powerful as to take on human traits. Or genetic engineers are able to grow a computer operator personality.

The thing to strive for above all in interface design is to capture the essence of a metaphor. Information

transfer, with the aid of a metaphor, is transformed from a serial data stream into a broad band flood. The user learns by analogy instead of by rote. He is also more likely to remember because he is building on his own experience, and the mental links seem to be much stronger [Ref. 4].

V. AMIABLE UNIX

As originally conceived UNIX was a research tool designed to run on a small computer in a laboratory. It became popular because it was easy to use, versatile and readily expandable. Now that it has been in existence for fifteen or more years, the only feature it shares with its first versions is ease of expansion. BELL Laboratories' baby is no longer easy to learn, and to use the wonderful facilities requires a moderately powerful machine. Amiable is a overlay for UNIX that conceals specifics of the operating system and allows a new user to use the system without first being intimate.

Amiable is implemented on a SUN model 150 work station, running Berkeley UNIX version 4.2, under SUN release version 1.1. The system is part of a research project for the Naval Electronics Systems Command, Command and Control Work Station.

Amiable consists of three major parts, a welcoming screen, the working environment, and an on-line VI screen editor help program. The work station is arranged so that the monochrome monitor is the text work area and the color monitor is for prompting menus and graphics work area. Each part will described and explained. The final section discusses future work.

A. WELCOME TO SUN UNIX

The Welcome package contains several program modules that display menus on the Sun color monitor. The screen choices are made with the Sun mouse, placed to the immediate right of the Sun keyboard. The entry menu presents the user with six choices and four short notes. The menu is in color on a dark background.

1. Main Menu Notes

The notes are very simple and to the point. The first note tells the user how to select using the mouse. The second tells him how to logout. The third and fourth are implementation notes, and a point of contact for more information.

2. Main Menu Selections

Of the six selections on the main menu, three are functional and two are under development. The sixth selection, actually number four, a programming environment, is just a title. The operational selections are:

1) Demonstrations.

2) Amiable UNIX

3) The C Shell

A program to select graphical attributes with the mouse is being developed for future inclusion. The graphics editor demonstration program is choice number six. It was included as an interim graphics editor, until it can either be upgraded or a new one written.

The main screen is simple and uncluttered. When a choice is made the main menu clears, replaced by either a demonstration menu or an attribute menu. Amiable and C shell selections clear the main menu but leave the welcome screen on.

3. Demonstrations

Selected demonstrations may be chosen from this menu. The upper section is for monochrome selections, the lower section for color. The demonstrations are from those supplied with the machine, and are representative of system capability.

4. C Shell

This selection is nothing more than an escape into unadorned UNIX.

5. Amiable

The second selection is the Unix overlay Amiable. It is a shell script that helps the user do several things in Unix without any system knowledge, other than some generic computer experience. A C program version is in development; but is not yet reliable enough to unleash on the public.

B. INSIDE AMIABLE

Amiable is a mask for Unix. It incorporates the interface design ideas discussed in Chapters II through IV. It was written and implemented on the SUN, before the windowing software was installed, so it does not incorporate windowing, unfortunately. It does dramatically decrease the

amount of knowledge the user must bring with him to the Unix operating system. In most cases Amiable executes commands, the user supplies only file names. It has a terse toggle. When terse is chosen all messages from Amiable prompt for input only. Explanatory remarks and more complete requests for input are displayed with the prompts when terse is not set.

Amiable greets the user with a list of functions:

Browse Compile Edit Learn Move Print Shell eXecute loGout

The user can choose any of the functions by typing the capitalized letter in each. He may only choose one function at a time, but he may exit that function, leaving it running in the background, and choose another. Compile and Print use that feature. The idle state is with the function list displayed waiting for a user choice. Amiable asks the user for additional information needed for correct command execution.

The edit command causes the VI editor to be invoked on the monochrome monitor. It also brings up a program called "vicom" on the color monitor. Vicom is a pair of on-line help and tutorial screens, displayed while the user is in Edit. Vicom is intended to help the user learn and remember the VI editor.

1. Vicom

Vicom is a mouse driven help file. All selections within it are done with the mouse. It is two color screens containing VI editor commands and syntax. The first screen presented has five general information notes about the screens and usage. A second miscellaneous notes section is included on the first page of commands. They are notes about syntax notation used on the screen and some precautions about using the mouse.

Command meanings and syntax are presented in functional groups. The first page contains groups for: cursor control, text insertion, scrolling the screen, searching for strings, saving and ending a session. The second page has more advanced functional groups: moving text, using the buffers, and EX ':' commands. Within each functional group one or two basic general purpose commands are highlighted in a different color. New users are advised in the general information note to learn them first.

Each functional group is independent of the others and may be displayed by itself. Within each functional group are subgroups, information messages about the subgroups may be called by pointing the mouse at a command and pressing the left button. The ten or fewer lines of explanation are displayed just below screen center. The explanation contains usage notes and additional syntax rules, if any, as well as a reference to system documentation.

Two exercise files are on the system: VI_ex1, and VI_ex2. They are designed to give a new user more information about VI and some practice using the commands. A VI expert could do the simple exercises in less than thirty seconds, using the quickest methods. A new user could take half an hour or more. The lessons are not difficult; but the second one has several text editing operations that will keep him busy looking at the command screens.

C. TESTING

Amiable is essentially untested; time ran out. Criticisms and comments from six colleagues who used the system briefly have been incorporated. The system is ready now for some exposure to a less well prepared group. The results of that trial by fire will not only tell whether change is needed, but how much. The rewrite is inevitable.

VI. CONCLUSIONS

Designing user interfaces is easy. Implementing the design to accommodate the foibles of human beings is hard. Literature on the topic of man-machine interface design abounds in trade journals, textbooks, hobby magazines, ... literally everywhere. Each author has a new wonderful plan to make it easy to use computers. There are only a handful of authors who speak from experience. Unless an author has credentials as a proven interface designer, who actually worked on the implementation, his ideas are suspect. Given time and manpower anything can be coded; but making a coherent package that works together is a different problem.

A. PEOPLE ARE STRANGE

The biggest stumbling block is people. They are unpredictable. Differences between two groups of people that do the same work in two different offices can stop an interface design in its tracks. And none of the faults will be discovered until the testing starts. Heckel has arrived at the same conclusions [Ref. 9]. The answer is to get prototypes running as early as possible and get it to the users. Solicit their comments, criticisms and suggestions. Then do something about them.

B. DESIGN MAINTAINABILITY IN

An interface must be designed to be changeable. Use every modern software maintenance technique to make the programs easy to change. Modularity is a key here. If a screen is a module, then modifying the screen is relatively easy. Within the screen module make the layout one or more modules, each menu should be one sub-module. Document, document, document! Assume no one from the original project will be around when the changes must be made. Explain everything, then the new people will only take half the time it took the original group to become familiar with the material.

C. DIFFERENT DRUMMERS

The group that sets out to design a user interface should have members from disciplines outside computer science. Ideally the group should at least have access to a cognitive psychologist, someone with a visual arts background (painter, graphic artist, photographer, cinematographer, etc.), and representative users.

An interface design implementation is too large a project for a single person, but the group must be responsive and flexible. Divide the project among several members (three to six) and schedule twice as much development time as is thought to be necessary. Test at every opportunity, then rewrite and retest.

Real human-computer interface packages are never finished, an euphemism for final product is: "Release Version XX".

D. BOOTSTRAP DESIGN

The current method of programming screens is not an effective use of resources. Too much time is spent getting simple menus, and their associated selection modules on the screen and running. A program or graphics editor should be able to handle that task with some development effort.

A graphics editor is the second logical choice for a workstation after a word processor. The next project on the SUN work station should be a graphics editor capable of drawing executable user interface screens.

The visible result of an interface design project in no way represents the amount of effort expended. A methodology must be developed to speed the process. Present methods are too hit or miss.

```
/*  att_choice.c                                        28 Mar 84
        This C program offers several selections based on attrib_menu.

        This routine is called by choice1 after attrib_menu has been
        placed on the screen.
                                                                    */

#include <usercore.h>
#include <stdio.h>

#define ATT_MENU 5

static float boxdx[] = {0.0,0.0,3.0,0.0,-3.0};
static float boxdy[] = {0.0,4.0,0.0,-4.0,0.0};

att_choice()
    {
     int choice, segname, pickid, done, att_menu;
     short i;

/*  This is case construct, is used to initiate the users choice
        of attributes.
*/
     set_segment_visibility(ATT_MENU, TRUE);
     done = FALSE;
     set_echo(LOCATOR,1,1);
     set_text_index(5);
     while (!done) {
       await_pick(0x00000000,1,&segname,&pickid);
         switch(pickid) {
         case 1: /*    Set Text Color */
           text("First choice");
         break;
         case 2: /*    Set Hilite Color */
           text("Second choice");
```

```
            break;
            case 3: /*    Set Header Color */
              text("Third choice");
            break;
            case 4: /*    Set Line Color   */
              text("Fourth choice");
            break;
            case 5: /*    Set Polygon Fill Color */
              text("Fifth choice");
            break;
            case 6: /*    Set Text Font  */
              text("Sixth choice");
            break;
            case 7: /*    Set Font Size  */
              text("Seventh choice");
            break;
            default: done = TRUE;
                     text("MOUSE choice");
                     break;
            } /* End of the "mnemonic_label" switch  */
          } /* End of while !done */
          set_segment_visibility(ATT_MENU, FALSE);
        } /* End of att_choice subroutine */

        /*****************************************************************************/
```

```
/*****************************************************************************/

/*    attrib_menu.c                          28 May 84

     This program was changed from menu1.c.  It is now the attribute
         change menu for setting text, color, and line attributes.

         It isa called by choice1 when the user select the Attributes
           choice.
         It does not call any other program segment. It is just a menu.
*/
#include <stdio.h>
#include <usercore.h>


static float dx[] = {0.0,0.0,40.0,0.0,-40.0};
static float dy[] = {0.0,-50.0,0.0,50.0,0.0};
static float sqrdx[] = {0.0,0.0,3.0,0.0,-3.0};
static float sqrdy[] = {0.0,-3.0,0.0,3.0,0.0};

attrib_menu(xpos,ypos)  /* menu selections for the main screen */
  float xpos,ypos;
  {
     /* Internal declarations section          */
     int inchar;
     short j, k, i = 0;
     float vsubsp = -3.0;
     float voptsp = -2.5;

     /*  Draw the box the menu will be contained in    */

      set_linewidth(0.3);
      move_abs_2(xpos,ypos);
      set_fill_index(1);
      move_abs_2(xpos,ypos);
```

```
        set_line_index(9);
        set_text_index(3);
        polyline_rel_2(dx,dy,5);

       /*  Insert the menu text    */

        move_rel_2(1.0,-2.0);
        text("ATTRIBUTE MENU");  /* Title of menu */
        move_rel_2(0.0,voptsp);
/* A vertical separator */
        line_rel_2(23.0,0.0);
/* Move for initial heading */
        move_rel_2(-23.0,vsubsp);

/* The following subtitle and options block may be reproduced several times
      within a menu - to save typing the skeleton repeatedly.
      */
        text("Color Combinations");
/* Move for first line of text */
        move_rel_2(0.0,vsubsp);
/*  The following are individual lines of text, and moves, etc. */
        set_pick_id(1);
        text("1.  Text Color");
        move_rel_2(0.0,voptsp);
        set_pick_id(2);
        text("2.  Hilite Color");
        move_rel_2(0.0,voptsp);
        set_pick_id(3);
        text("3.  Header Color");
        move_rel_2(0.0,voptsp);
        set_pick_id(4);
        text("4.  Line Color");
        move_rel_2(0.0,voptsp);
        set_pick_id(5);
        text("5.  Polygon Fill Color");
```

```
        move_rel_2(0.0,voptsp);

        set_pick_id(6);
        text("6.  Text Font");
        move_rel_2(0.0,vsubsp);
        set_pick_id(7);
        text("7.  Font Size");
        move_rel_2(0.0,voptsp);
        line_rel_2(38.0,0.0);
        move_rel_2(-38.0,vsubsp);
/* End of the subtitle and options block  */
        set_linewidth(0.0);
    /* Set Up the Color Table Selections*/
      k = 0;
      move_abs_2(-48.0,72.0);

      for (j=1;j<=3;j++) {
         for (i=1;i<13;i++) {
            set_pick_id(k+10);
            set_fill_index(7*k++);
            polygon_rel_2(sqrdx,sqrdy,5);
            move_rel_2(0.0,-4.0);
            }
         move_rel_2(4.0,48.0);
      }
}

/*****************************************************************************/
```

```
/*****************************************************************************/

/* This C program offers several selections based on menu1.

        The menu selections will consist of the standard options:

                1) Demonstrations.
                2) Run a UNIX Tutorial
                3) Enter a C Shell.
                                                                          */

#include <usercore.h>
#include <stdio.h>

#define MAIN_MENU 1
#define DEMO_MENU 2
#define ATT_MENU  5
#define WELCOME   6

static float redval[] = {0.0,1.0,.1,.2,.3,.4,.5,.6,.7,.8,.9,1.0,0.0};
static float grn[] = {0.0,1.0,.5,.6,.7,.8,.9,.99,.1,.2,.3,.4,1.0};
static float blu[] = {0.0,1.0,.9,.8,.7,.6,.5,.4,.3,.2,.1,0.0,1.0};
static float boxdx[] = {0.0,0.0,3.0,0.0,-3.0};
static float boxdy[] = {0.0,4.0,0.0,-4.0,0.0};
int cgldd(), bwldd();

extern struct vwsurf viewsurf;

choice1()
   {
     int choice, segname, pickid, done, att_menu;
     short i;

/*  This is case construct, is used to initiate the users choice
        of programs or other things he wishes to do.  It is the
        top level choice mechanism.                               */
```

```
define_color_indices(&viewsurf,0,12,redval,grn,blu);
create_temporary_segment();

att_menu = FALSE;
done = FALSE;
set_echo(LOCATOR,1,1);
set_text_index(5);
while (!done) {
  await_pick(1000000000,1,&segname,&pickid);
    switch(pickid) {
    case 1:
      set_segment_visibility(MAIN_MENU, FALSE);
      demo_choice();
      set_segment_visibility(MAIN_MENU, TRUE);
    break;
    case 2:
      set_segment_visibility(MAIN_MENU, FALSE);
      set_segment_visibility(WELCOME, FALSE);
      system("/usr/groen/env");
      new_frame();
      set_segment_visibility(WELCOME, TRUE);
      set_segment_visibility(MAIN_MENU, TRUE);
    break;
    case 3:
      set_segment_visibility(MAIN_MENU, FALSE);
      system("csh");
      set_segment_visibility(MAIN_MENU, TRUE);
    break;
    case 4:
      set_segment_visibility(MAIN_MENU, FALSE);
      text("Fourth choice ");
      set_segment_visibility(MAIN_MENU, TRUE);
    break;
    case 5:
      set_segment_visibility(MAIN_MENU, FALSE);
```

```
            if (!att_menu)  {
               close_temporary_segment();
               create_retained_segment(ATT_MENU);
               attrib_menu(-22.0,65.0);
               close_retained_segment();
               set_segment_detectability(ATT_MENU, TRUE);
               att_menu = TRUE;
               create_temporary_segment();
            }
            att_choice();
            set_segment_visibility(MAIN_MENU, TRUE);
          break;
          case 6:
            close_temporary_segment();
            set_segment_visibility(MAIN_MENU, FALSE);
            set_segment_visibility(WELCOME, FALSE);
            system( "/usr/demo/draw" );
            define_color_indices(&viewsurf,0,12,redval,grn,blu);
            new_frame();
            set_segment_visibility(WELCOME, TRUE);
            set_segment_visibility(MAIN_MENU, TRUE);
            create_temporary_segment();
          break;
          default: done = TRUE;
                   text("MOUSE choice");
                   break;
          } /* End of the "mnemonic_label" switch  */
      } /* End of while !done */
      close_temporary_segment();
    } /* End of choice1 subroutine */

    /*************************************************************************/
```

```
/*****************************************************************************/

/* This C program offers several selections based on demo_menu.
  It was copied from choice1.c on 27 May 84.

        The menu selections will consist of the selections from the
    demonstration directory:

                1) Monochrome demos.
                2) Color demos
                                                                        */

#include <usercore.h>
#include <stdio.h>

#define MAIN_MENU 1
#define DEMO_MENU 2

static float boxdx[] = {0.0,0.0,3.0,0.0,-3.0};
static float boxdy[] = {0.0,4.0,0.0,-4.0,0.0};

demo_choice()
   {
    int choice, segname, pickid, done;
    short i;

/*  This is case construct, is used to initiate the users choice
        of programs or other things he wishes to do.  It is the
        top level choice mechanism.                                     */

    done = FALSE;
    set_segment_visibility(DEMO_MENU, TRUE);
    set_echo(LOCATOR,1,1);
    set_text_index(5);
    while (!done){
    await_pick(120000000,1,&segname,&pickid);
```

```
        switch(pickid) {
            case 1:
              system("/usr/demo/stringart -d /dev/bwone0");
            break;
            case 2:
              system("/usr/demo/showmap");
            break;
            case 3:
              system("/usr/demo/rotcube");
            break;
            case 4:
              system("/usr/demo/chessgame -d /dev/twone0");
            break;
            case 5:
              system("/usr/demo/mixcolor -d /dev/cgone0");
            break;
            case 6:
              system("/usr/demo/product -d /dev/cgone0");
            break;
            case 7:
              system("/usr/demo/shaded /usr/demo/DATA/space.dat -d /dev/cgone0");
            break;
            case 8:
              system("/usr/demo/suncute -d /dev/cgone0");
            break;
            default:
                      done = TRUE;
                      break;
          } /* End of the switch  */
        } /* End of while !done */
        set_segment_visibility(DEMO_MENU, FALSE);
      } /* End of demo_choice subroutine */

      /*************************************************************************/
```

```
/*****************************************************************************/

/*  This program is a menu layout program for the demo frame.
        It is executed in main_choice, but remains invisible.
        Its visibility is controlled by the visibility attribute.
        It is visible when demo_choice is active.
        It calls:
                1.  Some number of Monochrome demo_menus
                2.  Some number of Color demo_menus
 */

#include <stdio.h>
#include <usercore.h>


static float dx[] = {0.0,0.0,40.0,0.0,-40.0};
static float dy[] = {0.0,-50.0,0.0,50.0,0.0};

demo_menu(xpos,ypos)  /* menu selections for the main screen */
  float xpos,ypos;
  {
     /* Internal declarations section          */
     int inchar;
     short i = 0;
     float vsubsp = -3.0;
     float voptsp = -2.5;

     /*  Draw the box the menu will be contained in    */

      set_linewidth(0.3);
      move_abs_2(xpos,ypos);
      set_fill_index(1);
      move_abs_2(xpos,ypos);
      set_line_index(9);
      set_text_index(3);
```

```
        polyline_rel_2(dx,dy,5);

       /*  Insert the menu text   */

        move_rel_2(1.0,-2.0);
        text("DEMONSTRATIONS");  /* Title of menu */
        move_rel_2(0.0,voptsp);
/* A vertical separator */
        line_rel_2(23.0,0.0);
/* Move for initial heading */
        move_rel_2(-23.0,vsubsp);

/* The following subtitle and options block may be reproduced several times
      within a menu - to save typing the skeleton repeatedly.
      */
        text("Monochrome");
/* Move for first line of text */
        move_rel_2(0.0,vsubsp);
/*  The following are individual lines of text, and moves, etc. */
        set_pick_id(1);
        text("1.  Stringart");
        move_rel_2(0.0,voptsp);
        set_pick_id(2);
        text("2.  Maps");
        move_rel_2(0.0,voptsp);
        set_pick_id(3);
        text("3.  Rotate");
        move_rel_2(0.0,voptsp);
        set_pick_id(4);
        text("4.  Chess");
        move_rel_2(0.0,voptsp);
        line_rel_2(38.0,0.0);
        move_rel_2(-38.0,vsubsp);
/* End of the subtitle and options block  */
```

```
        text("Color");
/* Move for first line of text */
        move_rel_2(0.0,vsubsp);
/*  The following are individual lines of text, and moves, etc. */
        set_pick_id(5);
        text("5.  Mix Colors");
        move_rel_2(2.0,voptsp);
        set_pick_id(6);
        text("6.  Workstation");
        move_rel_2(0.0,voptsp);
        set_pick_id(7);
        text("7.  Shuttle");
        move_rel_2(0.0,voptsp);
        set_pick_id(8);
        text("8.  SUNcube");
        move_rel_2(0.0,voptsp);
        line_rel_2(38.0,0.0);
        move_rel_2(-38.0,vsubsp);
/* End of the subtitle and options block  */
        set_linewidth(0.0);
          }

/*****************************************************************************/
```

```
/*****************************************************************************/

                                                        29 May 84
        menu1.c
        This program is a generic menu layout program.  It contains
        code for a "main menu" and a secondary menu.  Either can be
        drawn within a box.
 */

#include <stdio.h>
#include <usercore.h>


static float dx[] = {0.0,0.0,40.0,0.0,-40.0};
static float dy[] = {0.0,-50.0,0.0,50.0,0.0};

menu1(xpos,ypos)  /* menu selections for the main screen */
  float xpos,ypos;
  {
     /* Internal declarations section          */
     int inchar;
     short i = 0;
     float vsubsp = -3.0;
     float voptsp = -2.5;

     /*  Draw the box the menu will be contained in    */

      set_linewidth(0.5);
      move_abs_2(xpos,ypos);
      set_fill_index(1);
      set_line_index(7);
      set_text_index(2);
      polyline_rel_2(dx,dy,5);

     /*  Insert the menu text   */
```

```
        move_rel_2(1.0,-2.0);
        text("SUN Workstation Selections"); /* menu title */

        move_rel_2(0.0,vsubsp);
    /* A vertical separator */
        line_rel_2(38.0,0.0);
    /* Move for initial heading */
        move_rel_2(-38.0,vsubsp);

        /* The first set of functions  */
    /* Move for first line of text */
    /*  The following are individual lines of text, and moves, etc. */
        set_pick_id(1);
        text("1. Demonstrations");  /* The first set of functions  */
        move_rel_2(0.0,vsubsp);
        set_pick_id(2);
        text("2. Amiable UNIX*");
        move_rel_2(0.0,vsubsp);
        set_pick_id(3);
        text("3. The C shell");
        move_rel_2(0.0,vsubsp);

    /*  The second set of functions   */

        set_pick_id(4);
        text("4. Programming Environment#");
        move_rel_2(0.0,voptsp);
        set_pick_id(5);
        text("5. Set Attributes");
        move_rel_2(0.0,voptsp);
        set_pick_id(6);
        text("6. The DRAW Demo");
        move_rel_2(0.0,vsubsp);
        line_rel_2(38.0,0.0);
        move_rel_2(-38.0,vsubsp);
```

```
        text("Select:  LEFT Mouse Button");
        move_rel_2(0.0,vsubsp);
        text("To LOGOUT - QUIT");
        move_rel_2(0.0,vcptsp);
        line_rel_2(20.0,0.0);
        move_rel_2(-20.0,vsubsp);
        text("* Not Fully Implemented");
        move_rel_2(0.0,vsubsp);
        text("# Not Even Started, would");
        move_rel_2(0.0,vsubsp);
        text("   you like to do it?");
        move_rel_2(2.0,vsubsp);
        text("   See Prof. RAHE.");
/* End of the subtitle and options block  */

        set_linewidth(0.0);
          }

/*****************************************************************************/
```

```
/*****************************************************************************

        welm.c                                                  29 May 84
This C program can be used by a novice SUN user to do some basic tasks on
the SUN system, especially the demonstrations.
It is the first screen that will be printed when a user logs into the system.
It offers several selections and entry levels ranging from first time user
to Kernigan or Richie.

        The menu selections will consist of the standard options:

                1) Edit a file.
                2) Run a program
                3) Other things I think up.
                4) Etc.
                5) Demonstrations
*/

#include <usercore.h>
#include <stdio.h>

#define hilite   254
#define cyan      12
#define red       63
#define blue     191
#define green    127
#define yellow   254
#define white      1
#define black      0

#define MAIN_MENU   1
#define DEMO_MENU   2
#define WELCOME     6
```

```
/*
        GLOBAL Variables
 */
static float red1[] = {.9961};
static float grn1[] = {0.0};
static float blu1[] = {.9961};

static float big_charht = 1.5,big_charwd = 1.1;

static float screendx[] = {0.0,65.0,0.0,-65.0};
static float screendy[] = {80.0,0.0,-80.0,0.0};
static float scr1dx[] = {0.0,64.0,0.0,-64.0};
static float scr1dy[] = {79.0,0.0,-79.0,0.0};
static float boxdx[] = {0.0,0.0,3.0,0.0,-3.0};
static float boxdy[] = {0.0,4.0,0.0,-4.0,0.0};
static float sm_box[] = {0.0,0.0,7.0,0.0,-7.0};
static float sm_boy[] = {0.0,-4.0,0.0,4.0,0.0};
int cg1dd(), tw2dd();

struct vwsurf viewsurf = DEFAULT VWSURF(cg1dd);

/*****************************************************************************/

main()
   {
     int done, pickid, segname;
     short i;
     char choice;

     setcore();

/*  The MAIN MENU routine is just a collection of character strings
        that list the possible things to do.
 */
     set charprecision(CHARACTER);
```

```
set_font(ROMAN);
set_charsize(big_charwd,big_charht);
set_visibility(FALSE);  /* Create these items but don't display yet */

create_retained_segment(MAIN_MENU);
menu1(-22.0,65.0);
set_segment_detectability(MAIN_MENU, TRUE);
close_retained_segment();

create_retained_segment(DEMO_MENU);
demo_menu(-22.0,65.0);
set_segment_detectability(DEMO_MENU, TRUE);
close_retained_segment();

set_visibility(TRUE);

create_retained_segment(WELCOME);
set_text_index(blue - 10);


move_abs_2(-35.0,1.0);
polyline_rel_2(screendx,screendy,4);
move_abs_2(-34.5,1.5);
polyline_rel_2(scr1dx,scr1dy,4);
move_abs_2(-27.0,72.0);
text("Welcome to the Sun Workstation");
move_abs_2(40.0,4.0);
set_fill_index(yellow);
set_line_index(green - 15);
polygon_rel_2(sm_box,sm_boy,5);
polyline_rel_2(sm_box,sm_boy,5);
move_abs_2(41.0,2.0);
text("QUIT");
move_abs_2(-30.0,10.0);
```

```
    close_retained_segment();

    set_segment_visibility(MAIN_MENU, TRUE);


    choice1(); /*  Exit to the first choice routine  */

    terminate_device(KEYBOARD,1);
    deselect_view_surface(&viewsurf);
    terminate_core();
    system("exit");
  }


/*
        coregraph.c is the SUN core graphics setup package for the UNIX
        environment under development.  It does only the initial SUN Core
        setup.  It started life in vicomms.c; but was extracted to increase
        modularity and decrease compile delays.


        Modfications and enhancements:

03 May 84 1.  Pared down initialization of vicomms.c to this form.  This file
                will be the initialization file for the UNIX environemt.

19 June 84 2. Updated to Suncore version 1.1

            Comments here (with action dates,please).
                                                                        */


setcore()
  {
    if(initialize_core(DYNAMICC,SYNCHRONOUS,THREED))
      exit(1);
```

```
        if(initialize_view_surface(&viewsurf,FALSE))
          exit(2);
        if(select_view_surface(&viewsurf))
          exit(3);
        initialize_device(BUTTON,1);
        initialize_device(BUTTON,2);
        initialize_device(BUTTON,3);
        initialize_device(PICK,1);
        initialize_device(LOCATOR,1);
        initialize_device(KEYBOARD,1);

        set_keyboard(1,80," ",1);
        set_echo_position(LOCATOR,1,0.0,0.0);
        set_echo_surface(LOCATOR,1,&viewsurf);
        set_echo_surface(PICK,1,&viewsurf);
        set_echo_surface(KEYBOARD,1,&viewsurf);
        set_output_clipping(TRUE);
        set_window_clipping(FALSE);

        set_viewport_2(0.0,1.0,0.0,0.75);
        set_window(-50.0,50.0,0.0,75.0);

    /* Set the MOUSE color */
        define_color_indices(&viewsurf,255,255,red1,grn1,blu1);
    }
    /*****************************************************************************/
```

```
#
# This is an experimental C shell script for a programming environment
#
#  Things to be added:
#       1) A Help file
#       2) Pages of commands and explanations for:
#               A. The vi editor - preliminary version done 24 APR 84.
#               B. Browsing files with MORE
#               C. The Learn module
#               D. A page of C shell instruction and explanations.
#               E. A page of C language instruction, definitions, etc.
#       3) Some of the present functions could be expanded to allow setting
#               different options.
#
echo -n 'Terse? y/n '
set terse=$<
if ($terse == y || $terse == Y) then
    set terse=y
endif
start:
if ($terse != y) then
    echo ' '
    echo 'What do you want to do?:'
    echo ' '
endif
echo 'Browse Compile Edit Learn* Move Print Shell eXecute loGout'
if ($terse != y) then
    echo ' '
    echo 'To pick a function from above, type the letter that is capitalized'
    echo 'Functions marked * are not yet implemented.'
endif
set char=($<)
switch($char)

case b:
```

```
case F:
    echo "BROWSE Files and Directories"
browse:
    echo -n 'Do you want to look at a File or Directory?'
    set char=($<)
    if ($char != q && $char != Q) then
        if ($char == F || $char == f) then
            echo 'Which file?'
            echo 'Do not BROWSE binary files, an error results.'
            ls -F
            set file=$<
            if ($file != q && $file != Q)  then
                more -cds $file
            endif
        else echo -n 'Which directory? '
            ls -F
            set directry=$<
            if ($directry != q && $directry != Q)  then
                echo 'Long or Short version? '
                set vers=($<)
                if (($vers == l || $vers == L)&&($vers != q && $vers != Q)) then
                    ls -als $directry
                    if ($terse != y) then
                        echo "The command was 'ls -als $directry'."
                    endif
                else ls -F $directry
                if ($terse != y) then
                    echo "The command was 'ls -F $directry'."
                        endif
                    endif
                endif
            endif
            echo -n 'Continue Browsing? y/n '
            set char=($<)
            if (($char == y || $char == Y)&&($char != q && $char != Q)) goto browse
```

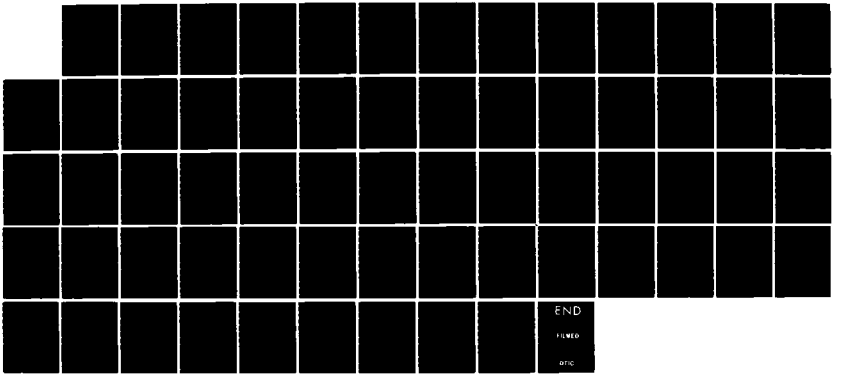
```
        endif # end of if char == q
        breaksw

    case c:
    case C:
        if ($terse != y) then
            echo -n 'COMPILE -  Did you just edit this file? y/n? '
        else echo -n 'COMPILE - Same file y/n? '
        endif
        set char=($<)
        if ($char != q && $char != Q) then
            if ($char == n || $char == N) then
                echo  'Which file do you want to compile?'
                if (terse != y)  then
                    ls -F
                endif
                set file=($<)
                if ($file == q || $file == Q) goto start
            endif
         #  if ($terse != y) then
         #      echo 'Do you want this file compiled as part of a MAKE? y/n? '
         #  else echo -n 'MAKE? y/n? '
         #  endif
         #  set char=($<)
         #  if (($char == y || $char == Y)&&($char != q && $char != Q)) then
         #      echo -n 'Which one?'
         #      set mkfile=($<)
         #      make -f $mkfile >&! comp_errs &
         #  else if (($char != y || $char != Y)&&($char != q && $char != Q)) then
                if ($terse != y) then
                    echo 'Which libraries do you want used in the compile?'
                else echo -n 'Libraries? '
                endif
                set libraries=($<)
                if ($libraries != q && $libraries != Q)  then
```

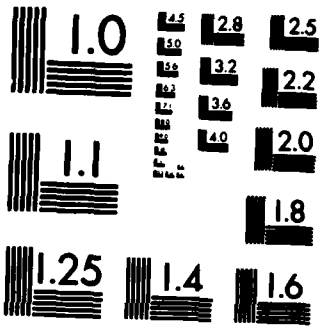
```
                cc $file $libraries >&! comp_errs &
            endif
        #  endif
        endif
        breaksw

    case e:
    case E:
        if ($terse != y) then
            echo 'EDIT - which file do you want to edit? '
            ls -F
        else echo 'EDIT - Filename? '
            ls -F
        endif
        set file=($<)
        if ($file != q && $file != Q)  then
            vcm &
            vi $file
        endif
        breaksw

    case g:
    case G:
        echo -n "LOGOUT - type lg to log out"
        exit
        breaksw


    case l:
    case L:
        echo "LEARN - Not yet implemented"
        breaksw

    case m:
    case M:
```

```
        echo  "MOVE - Which directory do you want to move to"
        if ($terse != y) then
            echo  "Type the whole pathname."
            ls -F
        endif
        set directry=$<
        if ($directry != q && $directry != Q)  then
            cd $directry
            pwd
            if ($terse != y) then
                echo "The command invoked was 'cd $directry'."
            endif
        endif
        breaksw

    case p:
    case P:
        echo 'PRINT - Type the names of the file(s) you want printed'
        echo 'Do not PRINT binary files, an error results.'
        if ($terse != y) then
            ls -F
        endif
        set file=($<)
        if ($file != q && $file != Q)  then
            pr $file | lpr
            if ($terse != y) then
                echo "The command invoked was 'pr $file | lpr'."
            endif
        endif
        breaksw

    case q:
    case Q:
        echo "QUIT"
        exit
```

```
        breaksw

    case s:
    case S:
        if ($terse != y) then
            echo 'SHELL - You are now in a C shell, type one line commands.'
            echo '        Type exit or ^D to terminate the shell.'
        endif
        csh -st
        breaksw

    case x:
    case X:
        echo "EXECUTE - Type the executable filename."
        set file=($<)
        if ($file != q && $file != Q)  then
            $file
        endif
        breaksw

    endsw

    goto start
```

AD-A150 558 A DESIGN ANALYSIS AND IMPLEMENTATION OF A USER-FRIENDLY 2/2
INTERFACE FOR THE UNIX OPERATING SYSTEM(U) NAVAL
POSTGRADUATE SCHOOL MONTEREY CA F E GROENERT JUN 84
UNCLASSIFIED F/G 9/2 NL





```
/*
        vicomms.c   10 April 84   F. Groenert
            last modified 3 May 84 by F. Groenert
            last modified 18 June 84 by F. Groenert
        The 7 June modification involved changing sunbitmap and suncolor
        to the new names for SUN release 1.1.  They are now:
                        bwidd for sunbitmap
                        cgidd for suncolor.

        A program written in "C", using Suncore graphics commands and
        subroutines to display a "working subset" of the VI editor commands.

        It is used within "env" to display a set of vi editor
        commands on the color monitor, while using that editor.  It
        does not display the complete set; but a complete enough set.

        Modfications and enhancements:
            Comments here (with action dates,please).                    */

#include <usercore.h>
/*
        GLOBAL Variables
 */
static float redtex[] = {0.0,1.0,.1,.2,.3,.4,.5,.6,.7,.8,.9,1.0,0.0};
static float grntex[] = {0.0,1.0,.5,.6,.7,.8,.9,.99,.1,.2,.3,.4,1.0};
static float blutex[] = {0.0,1.0,.9,.8,.7,.6,.5,.4,.3,.2,.1,0.0,1.0};
static float red1[] = {.9961};
static float grn1[] = {0.0};
static float blu1[] = {.9961};

static float screendx[] = {0.0,0.0,98.0,0.0,-98.0};
static float screendy[] = {0.0,73.0,0.0,-73.0,0.0};
static float scr1dx[] = {0.0,0.0,97.0,0.0,-97.0};
static float scr1dy[] = {0.0,72.0,0.0,-72.0,0.0};
static float boxdx[] = {0.0,0.0,3.0,0.0,-3.0};
```

```
static float tcxdy[] = {0.0,4.0,0.0,-4.0,0.0};
int twidd(),cgidd();

struct vwsurf viewsurf = DEFAULT_VWSURF(cgidd);

short hilite  = 120;
short cyan    = 12;
short red     = 63;
short blue    = 191;
short green   = 127;
short yellow  = 254;
short white   = 1;
short black   = 0;
short text_color   = 191;
short header_color = 63;

static float VI_messageX = -47.0;
static float VI_messageY = 27.0;

static float big_charht = 3.0,big_charwd = 2.2;
static float charht = 1.4,charwd = 1.0;
static float lf = -1.4;
static float tab = 3.0;
static float sp = 1.0;

static int VI_MENU    = 1;
static int CHOOSE     = 2;
static int CURS_MV    = 3;
static int INS_TEXT   = 4;
static int SCRN_MV    = 5;
static int FIND       = 6;
static int FINISH     = 7;
static int MISC       = 8;
static int ERASE      = 9;
static int CURS_MSG1      = 10;
```

```
static int CURS_MSG2       = 11;
static int CURS_MSG3       = 12;
static int CURS_MSG4       = 13;
static int FIND_MSG        = 14;
static int INS_TEXT_MSG1   = 15;
static int INS_TEXT_MSG2   = 16;
static int INS_TEXT_MSG3   = 17;
static int INS_TEXT_MSG4   = 18;
static int INS_TEXT_MSG5   = 19;
static int INS_TEXT_MSG6   = 20;
static int INS_TEXT_MSG7   = 21;
static int INS_TEXT_MSG8   = 22;
static int INS_TEXT_MSG9   = 23;
static int SCRN_MV_MSG1    = 24;
static int SCRN_MV_MSG2    = 25;
static int SCRN_MV_MSG3    = 26;
static int FINISH_MSG1     = 27;
static int FINISH_MSG2     = 28;
static int COLON_CMDS      = 29;
static int MOVE_TEXT       = 30;
static int BUFFERS         = 31;
static int GEN_INFO        = 32;
static int BUFFER_MSG1     = 33;
static int BUFFER_MSG2     = 34;
static int BUFFER_MSG3     = 35;
static int MOVE_TEXT_MSG1  = 36;
static int MOVE_TEXT_MSG2  = 37;
static int MOVE_TEXT_MSG3  = 38;
static int COLON_MSG1      = 39;
static int COLON_MSG2      = 40;
static int COLON_MSG3      = 41;
static int COLON_MSG4      = 42;
static int COLON_MSG5      = 43;
static int COLON_MSG6      = 44;
static int COLON_MSG7      = 45;
```

```
/*****************************************************************************

        The main body.  Within the main body the Suncore is initialized
and all the segments are constructed prior to the choice mechanism.
The choice mechanism is the switch construct.  When a segment is picked
with the mouse, values for both the segment and the pickid are returned.
The first stage of selection is based on the segment name chosen.  The
following selections are a switch case, based on the pickid number.  The
individual segments are made to appear and disappear by turning their
visibility attribute on and off.
                                                                        */

main(argc,argv)
    int argc;
    char argv[];
    {
      short i, cursor_move, find_seg, ins_txt, scrn_mv_seg, finish_seg;
      short buffer_seg, colon_seg, move_text_seg;
      int butnum,done,pickid,segname,c;
      float xmax, ymax, x, y, xmin, ymin;

/*
        Set-up the SUNCore graphics package
                                                                        */

      if(initialize_core(DYNAMICC,SYNCHRONOUS,TREED))
        exit(1);
      if(initialize_view_surface(&viewsurf,FALSE))
        exit(2);
      define_color_indices(&viewsurf,0,12,redtex,grntex,blutex);
      if(select_view_surface(&viewsurf))
        exit(3);
      initialize_device(BUTTON,1);
      initialize_device(BUTTON,3);
      initialize_device(PICK,1);
```

```
        initialize_device(LOCATOR,1);
        initialize_device(KEYBOARD,1);
        set_keyboard(1,80,    ,1);
        set_echo_surface(LOCATOR,1,&viewsurf);
        set_echo_surface(PICK,1,&viewsurf);
        set_echo_surface(KEYBOARD,1,&viewsurf);
        set_output_clipping(TRUE);
        set_window_clipping(FALSE);

        set_viewport_2(0.0,1.0,0.0,0.75);
        set_window(-50.0,50.0,0.0,75.0);
        set_charsize(big_charwd,big_charht);

        define_color_indices(&viewsurf,255,255,red1,grn1,blu1);

    /*      The next block of statements set the default colors for
            various categories of text.  If different colors are
            specified ay the calling program they are substituted
            here.                                                    */
    /*
            header_color = red;
            text_color = cyan;
            if (argc > 1) {
                if (argv[1] != ' ') header_color = argv[1];
                if (argv[2] != ' ') text_color = argv[2];
            } */

    /*  The next three lines are static color selections for the screen outline
            and title.                                               */

        set_text_index(blue);
        set_line_index(green);
        set_fill_index(black);

     /*
```

```
        Create the background screen segment for the VI commands.
                                                                    */

/*    setup_vicom();     A routine that initializes SUNCore */
/*    VI_screen1();      Sets up the initial screen segment */
      create_retained_segment(VI_MENU);
      set_charprecision(CHARACTER);
      set_font(ROMAN);
      move_abs_2(-49.0,1.0);
      polygon_rel_2(screendx,screendy,5);
      polyline_rel_2(screendx,screendy,5);
      move_abs_2(-48.5,1.5);
      polyline_rel_2(scridx,scridy,5);
      move_abs_2(-32.0,71.0);
      text("VI Editor Commands");
      close_retained_segment();

/*

        The next series of calls are the routines that make the individual
    segments.  Once the segment's call has run, that segment's appearance is
    controlled by its visibility attribute.

*/
      set_charsize(charwd,charht);
      set_charprecision(STRING);  /* Reset the text characteristics to */
      set_font(SCRIPT);           /* a faster writing font.            */
      choose(-47.0,4.4);
      erase(VI_messageX,VI_messageY+0.5);
      gen_info(VI_messageX,VI_messageY+8.0);
/*  a greeting screen will go here  */
      set_visibility(FALSE);
      curs_mv(-47.0,69.0);
      scrn_mv(3.0,69.0);
      ins_text(-47.0,51.5);
```

```
        find(3.0,48.0);
        finish(3.0,39.2);
        misc(VI_messageX+sp,VI_messageY);
        colon_cmds(-47.0,69.0);
        move_text(3.0,69.0);
        buffers(-47.0,50.0);

    /*
        This is where the action takes place.  All previous code was
      initialization.  In here the mouse is used as a pick to choose the
      subject for further information.  Layers of information are arranged
      so that cursory memory joggers are on the surface; but more in-depth
      info is just a button press away.                                  */

        while (!done) {
            set_echo(LOCATOR,1,1);
            await_pick(1000200600,1,&segname,&pickid);
            if (segname == CHOOSE) {
                set_segment_visibility(ERASE,FALSE);
                switch(pickid) {
                  case 1:               /* Display the cursor motion commands */
                        set_segment_visibility(COLON_CMDS, FALSE);
                        set_segment_visibility(CURS_MV, TRUE);
                        set_segment_visibility(INS_TEXT, FALSE);
                        set_segment_visibility(SCRN_MV, FALSE);
                        set_segment_visibility(FIND, FALSE);
                        set_segment_visibility(FINISH, FALSE);
                        set_segment_visibility(MISC, FALSE);
                        set_segment_visibility(MOVE_TEXT, FALSE);
                        set_segment_visibility(BUFFERS, FALSE);
                        set_segment_visibility(GEN_INFO, FALSE);
                        break;
                  case 2:               /* Display the text insertion commands */
                        set_segment_visibility(CURS_MV, FALSE);
```

```
            set_segment_visibility(BUFFERS, FALSE);
            set_segment_visibility(GEN_INFO, FALSE);
            set_segment_visibility(INS_TEXT, TRUE);
            set_segment_visibility(SCRN_MV, FALSE);
            set_segment_visibility(FIND, FALSE);
            set_segment_visibility(FINISH, FALSE);
            set_segment_visibility(MISC, FALSE);
            set_segment_visibility(COLON_CMDS, FALSE);
            set_segment_visibility(MOVE_TEXT, FALSE);
            break;
    case 3:          /* Display the screen motion commands */
            set_segment_visibility(CURS_MV, FALSE);
            set_segment_visibility(INS_TEXT, FALSE);
            set_segment_visibility(MOVE_TEXT, FALSE);
            set_segment_visibility(SCRN_MV, TRUE);
            set_segment_visibility(FIND, FALSE);
            set_segment_visibility(FINISH, FALSE);
            set_segment_visibility(MISC, FALSE);
            set_segment_visibility(COLON_CMDS, FALSE);
            set_segment_visibility(BUFFERS, FALSE);
            set_segment_visibility(GEN_INFO, FALSE);
            break;
    case 4:                  /* Display the Search commands */
            set_segment_visibility(CURS_MV, FALSE);
            set_segment_visibility(INS_TEXT, FALSE);
            set_segment_visibility(SCRN_MV, FALSE);
            set_segment_visibility(BUFFERS, FALSE);
            set_segment_visibility(FIND, TRUE);
            set_segment_visibility(FINISH, FALSE);
            set_segment_visibility(MISC, FALSE);
            set_segment_visibility(COLON_CMDS, FALSE);
            set_segment_visibility(MOVE_TEXT, FALSE);
            set_segment_visibility(GEN_INFO, FALSE);
            break;
    case 5:                  /* Display Exit and Save commands */
```

```
                set_segment_visibility(CURS_MV, FALSE);
                set_segment_visibility(INS_TEXT, FALSE);
                set_segment_visibility(SCRN_MV, FALSE);
                set_segment_visibility(FIND, FALSE);
                set_segment_visibility(GEN_INFO, FALSE);
                set_segment_visibility(FINISH, TRUE);
                set_segment_visibility(MISC, FALSE);
                set_segment_visibility(COLON_CMDS, FALSE);
                set_segment_visibility(MOVE_TEXT, FALSE);
                set_segment_visibility(BUFFERS, FALSE);
                break;
        case 6:                         /* Display all VI editor commands */
                set_segment_visibility(COLON_CMDS, FALSE);
                set_segment_visibility(MOVE_TEXT, FALSE);
                set_segment_visibility(BUFFERS, FALSE);
                set_segment_visibility(GEN_INFO, FALSE);
                set_segment_visibility(CURS_MV, TRUE);
                set_segment_visibility(INS_TEXT, TRUE);
                set_segment_visibility(SCRN_MV, TRUE);
                set_segment_visibility(FIND, TRUE);
                set_segment_visibility(FINISH, TRUE);
                set_segment_visibility(MISC, TRUE);
                break;
        case 7: done = TRUE;                    /* Quit this screen */
                break;
        case 8:                         /* Display Page 2 */
                set_segment_visibility(CURS_MV, FALSE);
                set_segment_visibility(INS_TEXT, FALSE);
                set_segment_visibility(SCRN_MV, FALSE);
                set_segment_visibility(FIND, FALSE);
                set_segment_visibility(FINISH, FALSE);
                set_segment_visibility(MISC, FALSE);
                set_segment_visibility(COLON_CMDS, TRUE);
                set_segment_visibility(MOVE_TEXT, TRUE);
                set_segment_visibility(BUFFERS, TRUE);
```

```
            set_segment_visibility(GEN_INFO, TRUE);
            break;
case 9:                         /* Display Colon commands */
            set_segment_visibility(CURS_MV, FALSE);
            set_segment_visibility(INS_TEXT, FALSE);
            set_segment_visibility(SCRN_MV, FALSE);
            set_segment_visibility(FIND, FALSE);
            set_segment_visibility(FINISH, FALSE);
            set_segment_visibility(MISC, FALSE);
            set_segment_visibility(COLON_CMDS, TRUE);
            set_segment_visibility(MOVE_TEXT, FALSE);
            set_segment_visibility(BUFFERS, FALSE);
            set_segment_visibility(GEN_INFO, FALSE);
            break;
case 10:                        /* Display Move Text commands */
            set_segment_visibility(CURS_MV, FALSE);
            set_segment_visibility(INS_TEXT, FALSE);
            set_segment_visibility(SCRN_MV, FALSE);
            set_segment_visibility(FIND, FALSE);
            set_segment_visibility(FINISH, FALSE);
            set_segment_visibility(MISC, FALSE);
            set_segment_visibility(COLON_CMDS, FALSE);
            set_segment_visibility(MOVE_TEXT, TRUE);
            set_segment_visibility(BUFFERS, FALSE);
            set_segment_visibility(GEN_INFO, FALSE);
            break;
case 11:                        /* Display Buffer Info and commands */
            set_segment_visibility(CURS_MV, FALSE);
            set_segment_visibility(INS_TEXT, FALSE);
            set_segment_visibility(SCRN_MV, FALSE);
            set_segment_visibility(FIND, FALSE);
            set_segment_visibility(FINISH, FALSE);
            set_segment_visibility(MISC, FALSE);
            set_segment_visibility(COLON_CMDS, FALSE);
            set_segment_visibility(MOVE_TEXT, FALSE);
```

```
                set_segment_visibility(BUFFERS, TRUE);
                set_segment_visibility(GEN_INFO, FALSE);
                break;
        case 12:                    /* Display General Info */
                set_segment_visibility(CURS_MV, FALSE);
                set_segment_visibility(INS_TEXT, FALSE);
                set_segment_visibility(SCRN_MV, FALSE);
                set_segment_visibility(FIND, FALSE);
                set_segment_visibility(FINISH, FALSE);
                set_segment_visibility(MISC, FALSE);
                set_segment_visibility(COLON_CMDS, FALSE);
                set_segment_visibility(MOVE_TEXT, FALSE);
                set_segment_visibility(BUFFERS, FALSE);
                set_segment_visibility(GEN_INFO, TRUE);
                break;
        default: done = TRUE;
           } /* End of switch for segname == CHOOSE. */
        } /* End of 'if (segname == CHOOSE) */

else if (segname == CURS_MV) {
  if (cursor_move == FALSE){
     VI_cursmsg1(); /* Establish these segments the first time  */
     VI_cursmsg2(); /* this segment is chosen. On subsequent    */
     VI_cursmsg3(); /* entries ignore this.                     */
     VI_cursmsg4();
     cursor_move = TRUE;
    } /* End of if(cursor_move == false) */

  set_segment_visibility(MISC,FALSE);
  set_segment_visibility(CURS_MSG1,FALSE); /* Reset cursor messages */
  set_segment_visibility(CURS_MSG2,FALSE);
  set_segment_visibility(CURS_MSG3,FALSE);
  set_segment_visibility(CURS_MSG4,FALSE);
  set_segment_visibility(ERASE,FALSE);  /* Ensure ERASE is reset so */
  set_segment_visibility(ERASE,TRUE);   /* it may be turned on      */
```

```
   switch (pickid) {
      case 1:
         set_segment_visibility(CURS_MSG1,TRUE);
              break;
      case 2:
      case 3:
      case 4:
         set_segment_visibility(CURS_MSG2,TRUE);
              break;
      case 5:
      case 6:
         set_segment_visibility(CURS_MSG3,TRUE);
              break;
      case 7:
      case 8:
      case 9:
         set_segment_visibility(CURS_MSG4,TRUE);
              break;
      default:
         set_segment_visibility(CURS_MV,FALSE);
    } /* End of switch(pickid) */
  }   /* End of If (segname == CURS_MV) */
else if (segname == ERASE)
   set_segment_visibility(ERASE,FALSE);

else if (segname == INS_TEXT) {
         if (ins_txt == FALSE) {
           Ins_text_msg1(-47.0,27.0);
           Ins_text_msg2(-47.0,27.0);
           Ins_text_msg3(-47.0,27.0);
           Ins_text_msg4(-47.0,27.0);
           Ins_text_msg5(-47.0,27.0);
           Ins_text_msg6(-47.0,27.0);
           Ins_text_msg7(-47.0,27.0);
           ins_txt = TRUE;
```

```
      } /* end of if(ins_txt ==FALSE) */
set_segment_visibility(MISC,FALSE);
set_segment_visibility(ERASE,FALSE);
set_segment_visibility(ERASE,TRUE);
      switch (pickid) {
        case 1:
              set_segment_visibility(INS_TEXT_MSG1,FALSE);
              set_segment_visibility(INS_TEXT_MSG1,TRUE);
              break;
        case 2:
        case 3:
              set_segment_visibility(INS_TEXT_MSG2,FALSE);
              set_segment_visibility(INS_TEXT_MSG2, TRUE);
              break;
        case 4:
        case 5:
              set_segment_visibility(INS_TEXT_MSG3,FALSE);
              set_segment_visibility(INS_TEXT_MSG3, TRUE);
              break;
        case 6:
              set_segment_visibility(INS_TEXT_MSG4,FALSE);
              set_segment_visibility(INS_TEXT_MSG4, TRUE);
              break;
        case 7:
        case 8:
              set_segment_visibility(INS_TEXT_MSG5,FALSE);
              set_segment_visibility(INS_TEXT_MSG5, TRUE);
              break;
        case 9:
              set_segment_visibility(INS_TEXT_MSG6,FALSE);
              set_segment_visibility(INS_TEXT_MSG6, TRUE);
              break;
        case 10:
        case 11:
        case 12:
```

```
                case 13:
                        set_segment_visibility(INS_TEXT_MSG7,FALSE);
                        set_segment_visibility(INS_TEXT_MSG7, TRUE);
                        break;
                default:
                        set_segment_visibility(ERASE,TRUE);
                        break;
              } /* End switch (pickid) in INS_TEXT. */
        } /* End of if(segname == INS_TEXT) */

      else if (segname == SCRN_MV) {
              if (scrn_mv_seg == FALSE) {
              Scrn_mv_msg1(VI_messageX,VI_messageY);
              Scrn_mv_msg2(VI_messageX,VI_messageY);
              Scrn_mv_msg3(VI_messageX,VI_messageY);
              scrn_mv_seg = TRUE;
              } /* End of if (scrn_mv_seg == FALSE) */
        set_segment_visibility(MISC,FALSE);
        set_segment_visibility(ERASE,FALSE);
        set_segment_visibility(ERASE,TRUE);
        switch(pickid) {
               case 1:
        set_segment_visibility(SCRN_MV_MSG1,FALSE); /* reset message, so */
        set_segment_visibility(SCRN_MV_MSG1,TRUE);  /* it will turn on. */
                  break;
               case 4:
               case 5:
               case 6:
               case 7:
               case 8:
               case 9:
                    set_segment_visibility(SCRN_MV_MSG2,FALSE);
                    set_segment_visibility(SCRN_MV_MSG2,TRUE);
                     break;
               case 10:
```

```
            case 11:
            case 12:
            case 13:
                 set_segment_visibility(SCRN_MV_MSG3,FALSE);
                 set_segment_visibility(SCRN_MV_MSG3,TRUE);
                   break;
            default:
                 set_segment_visibility(SCRN_MV, FALSE);
                   break;
       } /* End of switch(pickid) */
    } /* End of: else if (segname == SCRN_MV) */
else if (segname == FIND) {
        if (find_seg == FALSE) {
        find_msg(VI_messageX,VI_messageY);
        find_seg = TRUE;
        } /* End of if (find_seg == FALSE) */
  set_segment_visibility(MISC,FALSE);
  set_segment_visibility(ERASE,FALSE);
  set_segment_visibility(ERASE,TRUE);
  set_segment_visibility(FIND_MSG,FALSE); /* Ensure message is reset */
  set_segment_visibility(FIND_MSG,TRUE);  /* so it may be turned on. */
  } /* End of: else if (segname == FIND) */
else if (segname == FINISH) {
        if (finish_seg == FALSE) {
        Finish_msg1(VI_messageX,VI_messageY);
        Finish_msg2(VI_messageX,VI_messageY);
        finish_seg = TRUE;
        } /* End of if (finish_seg == FALSE) */
  set_segment_visibility(MISC,FALSE);
  set_segment_visibility(ERASE,FALSE);
  set_segment_visibility(ERASE,TRUE);
  switch(pickid) {
        case 1:                              /* Reset message so */
        case 2:                              /* it will turn on. */
          set_segment_visibility(FINISH_MSG1,FALSE);
```

```
                set_segment_visibility(FINISH_MSG1,TRUE);
                break;
              case 3:
              case 4:
              case 5:
                set_segment_visibility(FINISH_MSG2,FALSE);
                set_segment_visibility(FINISH_MSG2,TRUE);
                break;
              default:
                set_segment_visibility(FINISH, FALSE);
                break;
            } /* end of switch(pickid) */
          } /* End of: else if (segname == FINISH) */
        else if (segname == MOVE_TEXT) {
              if (move_text_seg == FALSE) {
              Move_text_msg1(VI_messageX,VI_messageY);
              Move_text_msg2(VI_messageX,VI_messageY);
              Move_text_msg3(VI_messageX,VI_messageY);
              move_text_seg = TRUE;
              } /* End of if (move_text_seg == FALSE) */
          set_segment_visibility(MISC,FALSE);
          set_segment_visibility(GEN_INFO,FALSE);
          set_segment_visibility(ERASE,FALSE);
          set_segment_visibility(ERASE,TRUE);
          switch(pickid) {
                case 1:                          /* Reset message so */
                case 2:                          /* it will turn on. */
                case 3:
                case 4:
                  set_segment_visibility(MOVE_TEXT_MSG1,FALSE);
                  set_segment_visibility(MOVE_TEXT_MSG1,TRUE);
                  break;
                case 5:
                case 6:
                case 7:
```

```
            set_segment_visibility(MOVE_TEXT_MSG2,FALSE);
            set_segment_visibility(MOVE_TEXT_MSG2,TRUE);
            break;
          case 8:
          case 9:
            set_segment_visibility(MOVE_TEXT_MSG3,FALSE);
            set_segment_visibility(MOVE_TEXT_MSG3,TRUE);
            break;
          default:
            set_segment_visibility(MOVE_TEXT, FALSE);
            break;
      } /* end of switch(pickid) */
    } /* End of: else if (segname == MOVE_TEXT) */
  else if (segname == BUFFERS) {
          if (buffer_seg == FALSE) {
          Buffer_msg1(VI_messageX,VI_messageY);
          Buffer_msg2(VI_messageX,VI_messageY);
          Buffer_msg3(VI_messageX,VI_messageY);
          buffer_seg = TRUE;
          } /* End of if (buffer_seg == FALSE) */
    set_segment_visibility(MISC,FALSE);
    set_segment_visibility(GEN_INFO,FALSE);
    set_segment_visibility(ERASE,FALSE);
    set_segment_visibility(ERASE,TRUE);
    switch(pickid) {
          case 1:                              /* Reset message so */
            set_segment_visibility(BUFFER_MSG1,FALSE);
            set_segment_visibility(BUFFER_MSG1,TRUE);
            break;                             /* it will turn on. */
          case 2:
          case 3:
            set_segment_visibility(BUFFER_MSG2,FALSE);
            set_segment_visibility(BUFFER_MSG2,TRUE);
            break;
          case 4:
```

```
                set_segment_visibility(BUFFER_MSG3,FALSE);
                set_segment_visibility(BUFFER_MSG3,TRUE);
                break;
              default:
                set_segment_visibility(BUFFERS, FALSE);
                break;
            } /* end of switch(pickid) */
          } /* End of: else if (segname == BUFFERS) */
        else if (segname == COLON_CMDS) {
              if (colon_seg == FALSE) {
              Colon_msg1(VI_messageX,VI_messageY);
              Colon_msg2(VI_messageX,VI_messageY);
              Colon_msg3(VI_messageX,VI_messageY);
              Colon_msg4(VI_messageX,VI_messageY);
              Colon_msg5(VI_messageX,VI_messageY);
              Colon_msg6(VI_messageX,VI_messageY);
              Colon_msg7(VI_messageX,VI_messageY);
              colon_seg = TRUE;
              } /* End of if (colon_seg == FALSE) */
          set_segment_visibility(MISC,FALSE);
          set_segment_visibility(GEN_INFO,FALSE);
          set_segment_visibility(ERASE,FALSE);
          set_segment_visibility(ERASE,TRUE);
          switch(pickid) {
                case 1:                              /* Reset message so */
                case 2:                              /* it will turn on. */
                  set_segment_visibility(COLON_MSG1,FALSE);
                  set_segment_visibility(COLON_MSG1,TRUE);
                  break;
                case 3:
                  set_segment_visibility(COLON_MSG2,FALSE);
                  set_segment_visibility(COLON_MSG2,TRUE);
                  break;
                case 4:
                case 5:
```

```
                    set_segment_visibility(COLON_MSG3,FALSE);
                    set_segment_visibility(COLON_MSG3,TRUE);
                    break;
                  case 6:
                    set_segment_visibility(COLON_MSG4,FALSE);
                    set_segment_visibility(COLON_MSG4,TRUE);
                    break;
                  case 7:
                  case 8:
                    set_segment_visibility(COLON_MSG5,FALSE);
                    set_segment_visibility(COLON_MSG5,TRUE);
                    break;
                  case 9:
                    set_segment_visibility(COLON_MSG6,FALSE);
                    set_segment_visibility(COLON_MSG6,TRUE);
                    break;
                  case 10:
                    set_segment_visibility(COLON_MSG7,FALSE);
                    set_segment_visibility(COLON_MSG7,TRUE);
                    break;
                  default:
                    set_segment_visibility(COLON_CMDS, FALSE);
                    break;
                } /* end of switch(pickid) */
              } /* End of: else if (segname == COLON_CMDS) */
            /* else {
              set_echo(LOCATOR,1,6);
               do {
               await_any_button_get_locator_2(1000000,1,&butnum,&xmax,&ymax);
               if (butnum == 1){
                 set_echo_position(LOCATOR,1,xmax,ymax);
                 xmin = xmax; ymin = ymax;
                 }
               } while (butnum != 3);
               if (xmax < xmin) {x=xmax; xmax=xmin; xmin=x;}
```

```
            if (ymax < ymin) {y=ymax; ymax=ymin; ymin=y;}
          } */
        }
        terminate_device(KEYBOARD,1);
        deselect_view_surface(&viewsurf);
    } /*  End of the main body of vicomms.c    */

/*
        The list of Cursor motion commands is constructed in curs_mv
                                                                    */

curs_mv(xpos,ypos)
  float xpos,ypos;
  {
    short i;

    create_retained_segment(CURS_MV);
    move_abs_2(xpos,ypos);
    set_text_index(header_color);
    text("CURSOR movement");
    set_text_index(hilite);
    move_rel_2(0.0,1f-0.5); set_pick_id(1);
    text("Single space - use arrow keys");
    set_text_index(text_color);
    move_rel_2(0.0,1f-0.5); set_pick_id(2);
    text("Top of screen.......H");
    move_rel_2(0.0,1f); set_pick_id(3);
    text("Bottom..............L");
    move_rel_2(0.0,1f); set_pick_id(4);
    text("Middle..............M");
    move_rel_2(0.0,1f-0.5); set_pick_id(5);
    text("End of line.........$");
    move_rel_2(0.0,1f); set_pick_id(6);
    text("Beginning...........0");
    move_rel_2(0.0,1f-0.5); set_pick_id(7);
```

```
    text("Forward one word....w,W");
    move_rel_2(0.0,lf); set_pick_id(8);
    text("Backward one word...b,B");
    move_rel_2(0.0,lf); set_pick_id(9);
    text("End of word.........e,E");
    move_rel_2(0.0,lf);
    close_retained_segment();
    set_segment_detectability(CURS_MV,TRUE);
    set_segment_visibility(CURS_MV,FALSE);
  }

/*
        The list of choices for the VI editor command types across the
  bottom of the screen is built in this routine.
                                                                        */

choose(xpos,ypos)
  float xpos,ypos;  /* These are the position for the list to be printed */
  {                 /* on the screen.  It may be changed in the calling  */
    short i;        /* program.                                          */

    create_retained_segment(CHOOSE);
     move_abs_2(xpos,ypos);
     set_text_index(text_color);
     text("SELECT>");
     move_rel_2(4.0*big_charwd+tab,0.0);
     set_text_index(header_color);
     set_pick_id(6);  text("Page 1:");
     set_text_index(header_color);
     move_rel_2(3.5*big_charwd+tab,0.0);
     set_pick_id(1);  text("Cursor");
     move_rel_2(3.0*big_charwd+tab,0.0);
     set_pick_id(2);  text("Insert/Delete");
     move_rel_2(7.5*big_charwd+tab,0.0);
     set_pick_id(3);  text("Screen");
```

```
        move_rel_2(3.5*big_charwd+tab,0.0);
        set_pick_id(4);  text("Find");
        move_rel_2(2.0*big_charwd+tab,0.0);
        set_pick_id(5);  text("Exit/Save");
        move_rel_2(5.0*big_charwd+tab,0.0);
        set_pick_id(7);  text("Quit");
        move_abs_2(xpos,ypos+lf);
        set_text_index(header_color);
        move_rel_2(4.0*big_charwd+tab,0.0);
        set_pick_id(8);  text("Page 2:");
        set_text_index(header_color);
        move_rel_2(3.5*big_charwd+tab,0.0);
        set_pick_id(9);  text("Colon-Commands");
        move_rel_2(8.5*big_charwd+tab,0.0);
        set_pick_id(10);  text("Move-Text");
        move_rel_2(6.0*big_charwd+tab,0.0);
        set_pick_id(11);  text("Buffers");
        move_rel_2(4.5*big_charwd+tab,0.0);
        set_pick_id(12);  text("Gen-Info");
        move_rel_2(2.0*big_charwd+tab,0.0);
        set_text_index(text_color);
      close_retained_segment();
      set_segment_detectability(CHOOSE,TRUE);
      set_segment_visibility(CHOOSE,TRUE);
    }

/*
        The initial greeting screen with explanations of how to get started
  and some rules of thumb goes in this position.
                                                                        */


/*
        The insert and delete text commands are listed in this routine.
                                                                        */
```

```
ins_text(xpos,ypos)
  float xpos,ypos;
  {
    short i;

    create_retained_segment(INS_TEXT);
    move_abs_2(xpos,ypos);
    set_text_index(header_color);
    text("INSERT or DELETE text:");
    move_rel_2(0.0,1f-0.5);
    set_text_index(hilite);
    set_pick_id(1);
    text("Before cursor...i,I..text<esc>");
    set_text_index(text_color);
    move_rel_2(0.0,1f);
    set_pick_id(2);
    text("After cursor....a,A..text<esc>");
    move_rel_2(0.0,1f);
    set_pick_id(3);
    text("Next line.......o,O..text<esc>");
    move_rel_2(0.0,1f-0.5);
    set_pick_id(4);
    text("Substitute.....(n)s..text<esc>");
    move_rel_2(0.0,1f);
    set_pick_id(5);
    text("Replace character.............r");
    move_rel_2(0.0,1f);
    set_pick_id(6);
    text("Join two lines................J");
    move_rel_2(0.0,1f);
    set_text_index(hilite);
    set_pick_id(7);
    text("Undo last change..............u");
    set_text_index(text_color);
    move_rel_2(0.3,1f);
```

```
        set_pick_id(8);
        text("Undo changes this line.......U");
        move_rel_2(0.0,lf-0.5);
        set_text_index(hilite);
        set_pick_id(9);
        text("Delete character(s).....(n)x,X");
        set_text_index(text_color);
        move_rel_2(0.0,lf-0.5);
        set_pick_id(10);
        text("Delete beginning of line....d^");
        move_rel_2(0.0,lf);
        set_pick_id(11);
        text("Delete end of line.........d$");
        move_rel_2(0.0,lf);
        set_pick_id(12);
        text("Delete word(s)...........(n)dw");
        move_rel_2(0.0,lf);
        set_text_index(hilite);
        set_pick_id(13);
        text("Delete line(s)...........(n)dd");
        set_text_index(text_color);
        move_rel_2(0.0,lf);
        close_retained_segment();
        set_segment_detectability(INS_TEXT,TRUE);
        set_segment_visibility(INS_TEXT,FALSE);
    }
/*
        The following DEFINE statements are for the INS_TEXT_MSG routines
                                                                        */

/*
        Text insertion command amplifying text is in this next routine.
  This entry covers the 'i' command and entry of special characters.
                                                                        */
Ins_text_msg1(xpos,ypos)
```

```
float xpos,ypos;
{
 create_retained_segment(INS_TEXT_MSG1);
 move_abs_2(xpos+sp,ypos);
 text("Insert Text: ");
 move_rel_2(0.0,1f);
 text("The command 'i' in VI is used to insert text before the present");
 move_rel_2(0.0,1f);
 text("cursor position.  It does not cause any text to be deleted or");
 move_rel_2(0.0,1f);
 text("overwritten.  'i' and similar commands must be used to open VI");
 move_rel_2(0.0,1f);
 text("for text input.  'I' causes text to be inserted at the beginning");
 move_rel_2(0.0,1f);
 text("of the current line.  The escape key <esc> must be depressed to");
 move_rel_2(0.0,1f);
 text("end the insert mode.");
 move_rel_2(0.0,2*1f);
 text("To enter an escape character in a VI file, preceed it with a ^V. ");
 move_rel_2(0.0,1f);
 text("The sequence ^V<esc> will print as: ^[.  This method works for ");
 move_rel_2(0.0,1f);
 text("all other characters that have special meaning in VI.");
 move_rel_2(0.0,1f);
 text("See sect. 3.1 in An Introduction to Display Editing with VI.");
 move_rel_2(0.0,1f);
 close_retained_segment();
 }  /* End of Ins_text_msg1 */

/*
       This procedure contains the entry for 'a' and 'o' .
                                                              */


Ins_text_msg2(xpos,ypos)
```

```
float xpos,ypos;
{
 create_retained_segment(INS_TEXT_MSG2);
 move_abs_2(xpos+sp,ypos);
 text("Append Text: ");
 move_rel_2(0.0,1f);
 text("The command 'a' in VI is used to append text after the present");
 move_rel_2(0.0,1f);
 text("cursor position.  It does not cause any text to be deleted or");
 move_rel_2(0.0,1f);
 text("overwritten.  'A' causes text to be inserted at the end");
 move_rel_2(0.0,1f);
 text("of the current line.  The escape key <esc> must be depressed to");
 move_rel_2(0.0,1f);
 text("end the append mode.");
 move_rel_2(0.0,1f);
 text(" The 'o' command is similar to 'i' and 'a'; but it causes the");
 move_rel_2(0.0,1f);
 text("input on the next line.  'O' puts the text on the previous line.");
 move_rel_2(0.0,1f);
 text("They are also terminated with <esc>.");
 move_rel_2(0.0,1f);
 close_retained_segment();
 }  /* End of Ins_text_msg2 */

/*
        This module contains the text entry for the 's' and 'r' text
  entry commands.                                                    */


Ins_text_msg3 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(INS_TEXT_MSG3);
    move_abs_2(xpos+sp,ypos);
```

```
        text("Substitute and Replace Text:");
        move_rel_2(0.0,1f);
        text("The 's' command causes text to be substituted for the character");
        move_rel_2(0.0,1f);
        text("under the cursor.  Preceeding the 's' with a number will cause");
        move_rel_2(0.0,1f);
        text("the substitution for that many characters.  'S' will cause");
        move_rel_2(0.0,1f);
        text("substitution for the entire line.  In any case the substitution");
        move_rel_2(0.0,1f);
        text("must be concluded with an <esc>.");
        move_rel_2(0.0,1f);
        text("  The 'r' command will replace the single character under the");
        move_rel_2(0.0,1f);
        text("cursor with the character of your choice.  'R' causes character");
        move_rel_2(0.0,1f);
        text("for character replacement until <esc>.");
        close_retained_segment();
      }  /* End of Ins_text_msg3 */

/*
        The text message for joining two lines is in here.
                                                          */

Ins_text_msg4 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(INS_TEXT_MSG4);
    move_abs_2(xpos+sp,ypos);
    text("Joining Lines:");
    move_rel_2(0.0,1f);
    text("To join the line the cursor is on and the next lower");
    move_rel_2(0.0,1f);
    text("line type 'J'.  The newly formed line will wrap to the next");
    move_rel_2(0.0,1f);
```

```
    text("line, if it is too long.  Enter an insert mode (i,a)");
    move_rel_2(0.0,1f);
    text("at the point where you want the line to end and type a <cr>.");
    move_rel_2(0.0,1f);
    close_retained_segment();
   }  /* End of Ins_text_msg4 */

/*
        This procedure is the text for the UNDO command
                                                                */

Ins_text_msg5 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(INS_TEXT_MSG5);
    move_abs_2(xpos+sp,ypos);
    text("Undoing changes:");
    move_rel_2(0.0,1f);
    text("The command 'u' will undo the last change you completed");
    move_rel_2(0.0,1f);
    text("   The 'U' command will undo all changes in the current line.");
    move_rel_2(0.0,1f);
    text("   These commands make use of the numbered buffers and thus");
    move_rel_2(0.0,1f);
    text("  can undo up to nine changes at once.  For more info ");
    move_rel_2(0.0,1f);
    text("   see sect 6.3 of An Introduction to Display Editing with VI .");
    close_retained_segment();
   }  /* End of Ins_text_msg5 */

/*
        This procedure contains the messages for the character
  deletion commands
                                                                */
```

```
Ins_text_msg5 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(INS_TEXT_MSG6);
    move_abs_2(xpos+sp,ypos);
    text("Deleting Characters:");
    move_rel_2(0.0,lf);
    text("The command 'x' will delete the character under the");
    move_rel_2(0.0,lf);
    text("cursor.  'X' will delete the character before the cursor.");
    move_rel_2(0.0,lf);
    text("Either may be prefaced with an integer denoting repetition.");
    move_rel_2(0.0,lf);
    text("Thus '6x' will delete 6 characters.  '6X' will delete six");
    move_rel_2(0.0,lf);
    text("characters before the cursor.");
    close_retained_segment();
   }  /* End of Ins_text_msg6 */

/*
        The following procedure contains the text for the delete
  commands and combinations therof.                                    */


Ins_text_msg7 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(INS_TEXT_MSG7);
    move_abs_2(xpos+sp,ypos);
    text("Deleting Groups of Words:");
    move_rel_2(0.0,lf);
    text("The commands 'd' (delete) and 'c' (change) are versatile ones ");
    move_rel_2(0.0,lf);
    text("  They are operators, of the form:");
    move_rel_2(0.0,lf);
```

```
        text("          (n)<op> <extent>");
        move_rel_2(0.0,1f);
        text("   (n)is an optional integer, meaning 'do n times'");
        move_rel_2(0.0,1f);
        text("   or 'operate on the n-th occurrence'.");
        move_rel_2(0.0,1f);
        text("<extent> uses characters already familiar: w,W,e,E,b,B,$,");
        move_rel_2(0.0,1f);
        text("   0(zero), ,/string/,?string?,<cr>(TWO lines!),n<cr>, etc.");
        move_rel_2(0.0,1f);
        text("When using the 'c' operator the last entry is the replacement");
        move_rel_2(0.0,1f);
        text("   string which must be ended with and <esc>.  It works like 's',");
        move_rel_2(0.0,1f);
        text("   but you don't have to count characters.  ");
        move_rel_2(0.0,1f);
        text("Doubling either - 'dd' or 'cc' means 'this entire line'.");
        move_rel_2(0.0,1f);
        text("See 3.3 - 4.4 of An Introduction to Display Editing with VI.");
        close_retained_segment();
      }  /* End of Ins_text_msg? */

/*
        Screen motion and control commands are listed here.
                                                              */

scrn_mv(xpos,ypos) /*  Screen control commands */
  float xpos,ypos;
  {
    short i;

    create_retained_segment(SCRN_MV);
    move_abs_2(xpos,ypos);
    set_text_index(header_color);
    text("SCREEN Movement");
```

```
move_rel_2(0.0,1f-0.5);
set_text_index(hilite);
set_pick_id(1);
text("Top of file.............1G");
set_text_index(text_color);
move_rel_2(0.0,1f);
set_pick_id(2);
text("Line #n.................nG");
set_text_index(hilite);
move_rel_2(0.0,1f);
set_pick_id(3);
text("Last line...............G");
move_rel_2(0.0,1f-0.5);
set_pick_id(4);
text("Forward screen(s)....(n)^F");
move_rel_2(0.0,1f);
set_pick_id(5);
text("Backward one screen.....^B");
set_text_index(text_color);
move_rel_2(0.0,1f);
set_pick_id(6);
text("Up half a screen........^U");
move_rel_2(0.0,1f);
set_pick_id(7);
text("Down half a screen......^D");
move_rel_2(0.0,1f);
set_pick_id(8);
text("One line at top.........^Y");
move_rel_2(0.0,1f);
set_pick_id(9);
text("One line at bottom......^E");
move_rel_2(0.0,1f);
set_pick_id(10);
text("This line mid screen.....z.");
move_rel_2(0.0,1f);
```

```
        set_pick_id(11);
        text("Previous location.....''");
        move_rel_2(0.0,1f);
        set_pick_id(12);
        text("Mark this slot.....m[a-z]");
        move_rel_2(0.0,1f);
        set_pick_id(13);
        text("Return to slot.....'[a-z]");
        move_rel_2(0.0,1f);
        close_retained_segment();
        set_segment_detectability(SCRN_MV,TRUE);
        set_segment_visibility(SCRN_MV,FALSE);
    }

/*
        Screen Movement commands using 'G'.                                    */

Scrn_mv_msg1 (xpos,ypos)
    float xpos,ypos;
    {
     create_retained_segment(SCRN_MV_MSG1);
     set_text_index(blue);
     move_abs_2(xpos+sp,ypos);
     text("Specific Line:");
     move_rel_2(0.0,1f);
     text("The 'G' command is used for moving to a specific line number.");
     move_rel_2(0.0,1f);
     text("  27G, causes line 27 to be at screen center.  To find the line");
     move_rel_2(0.0,1f);
     text("  numbers type ':set number'.  Line numbers will show at the left.");
     move_rel_2(0.0,1f);
     text("  Another way to get the same effect is ':<number>'.  See Colon ");
     move_rel_2(0.0,1f);
     text("  commands for more info.");
```

```
    move_rel_2(0.0,1f);
    text("See section 2.2 of An Introduction to Display Editing with VI .");
    close_retained_segment();
   }  /* End of Scrn_mv_msg1 */

/*
        Moving forward and backward thrcugh the file a screenfull at a time.
                                                                        */

Scrn_mv_msg2 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(SCRN_MV_MSG2);
    move_abs_2(xpos+sp,ypos);
    set_text_index(blue);
    text("Screen Forward or Backward:");
    move_rel_2(0.0,1f);
    text("'^F' or '^B' are used to move forward or backward by a screenfull.");
    move_rel_2(0.0,1f);
    text("'n^F' may be used to move forward 'n' screensfull.");
    move_rel_2(0.0,1f);
    set_text_index(header_color);
    text("WARNING!!");
    set_text_index(blue);
    move_rel_2(0.0,1f);
    text("'DO NOT attempt 'n^B' ; an editor bug may cause an infinite loop.");
    move_rel_2(0.0,1f);
    text("Try it at your own peril!");
    move_rel_2(0.0,1f);
    text("'^U' and '^D' move up or down a set number of lines, respectively.");
    move_rel_2(0.0,1f);
    text("You may specify that number by using it, i.e. 'n^D', after which");
    move_rel_2(0.0,1f);
    text("'^D' will always move that number of lines, until you type a new");
    move_rel_2(0.0,1f);
```

```
    text("number.");
    move_rel_2(0.0,1f);
    text("'^E' and '^Y' scroll a new line onto the screen bottom or top.");
    move_rel_2(0.0,1f);
    text("Either may be prefixed by a number: 'n^E' or 'n^Y'.");
    close_retained_segment();
  }  /* End of Scrn_mv_msg2 */

/*
        Line position control and Marking locations in the file.
                                                                */

Scrn_mv_msg3 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(SCRN_MV_MSG3);
    move_abs_2(xpos+sp,ypos);
    text("Move This Line:");
    move_rel_2(0.0,1f);
    text("'z<cr>' will move the line the cursor is on to the top line.");
    move_rel_2(0.0,1f);
    text("'z.' will move the line the cursor is on to mid-screen.");
    move_rel_2(0.0,1f);
    text("'z-' will move the line the cursor is on to the bottom line.");
    move_rel_2(0.0,2*1f);
    text("MARKERS:");
    move_rel_2(0.0,1f);
    text("Two single quote marks '' will move you to your previous position");
    move_rel_2(0.0,1f);
    text("in the file.  MARKERS allow you to mark particular spots in the");
    move_rel_2(0.0,1f);
    text("file, allowing you to jump easily from spot to spot. To mark a");
    move_rel_2(0.0,1f);
    text("spot: position the cursor on the desired line and type 'm<letter>'");
    move_rel_2(0.0,1f);
```

```
    text("To return to that spot type <single quote><letter>, e.g. 'a .");
    move_rel_2(0.0,1f);
    text("Use only lower case letters to mark positions.");
    close_retained_segment();
  }  /* End of Scrn_mv_msg3 */


/*
        The search commands are listed here.
                                                                    */

find(xpos,ypos)  /* Find a string in text. */
  float xpos,ypos;
  {
     short i;

     create_retained_segment(FIND);
     move_abs_2(xpos,ypos);
     set_text_index(header_color);
     text("FIND a String");
     move_rel_2(0.0,1f-0.5);
     set_text_index(hilite);
    set_pick_id(1);
     text("Search forwards...../string/<cr>");
     set_text_index(text_color);
     move_rel_2(0.0,1f);
    set_pick_id(2);
     text("Search backwards....?string?<cr>");
     move_rel_2(0.0,1f);
    set_pick_id(3);
     text("Repeat last search.........n,N");
     move_rel_2(0.0,1f);
    close_retained_segment();
    set_segment_detectability(FIND,TRUE);
    set_segment_visibility(FIND,FALSE);
```

```
    }

/*
        The following routine is the supplementary message for the searching
  commands.
                                                                        */

find_msg(xpos,ypos)
  float xpos,ypos;

  {
    create_retained_segment(FIND_MSG);
    move_abs_2(xpos+sp,ypos);
    text("Searching for Character Strings:");
    move_rel_2(0.0,lf);
    text("    Letting the computer find a given string in a file is easier");
    move_rel_2(0.0,lf);
    text("than trying to search for it yourself, especially if you must find");
    move_rel_2(0.0,lf);
    text("every occurrence.  The letter 'n' allows you to repeat the last");
    move_rel_2(0.0,lf);
    text("search, for the next occurrence of string.  'N' repeats the");
    move_rel_2(0.0,lf);
    text("search in the opposite direction.  '// and ?? followed by a <cr>");
    move_rel_2(0.0,lf);
    text("have a similar effect.");
    move_rel_2(0.0,lf);
    text("For `almost-global` changes the first occurrence can be found");
    move_rel_2(0.0,lf);
    text("using /.../ or ?...? and the change made; from then on 'n' (next");
    move_rel_2(0.0,lf);
    text("occurrence) and '.' (repeat change) are used for all subsequent");
    move_rel_2(0.0,lf);
    text("changes.");
    move_rel_2(0.0,lf);
```

```
    text("See sections 15-17, and 35 of the VI tutorial for more info and ");
    move_rel_2(0.0,1f);
    text("some very esoteric searching commands");
    close_retained_segment();
    set_segment_detectability(FIND_MSG,TRUE);
  }

/*
        The ways to leave a VI file are listed here.
                                                                */

finish(xpos,ypos)  /* Commands that exit the editor */
  float xpos,ypos;
  {
     short i;

    create_retained_segment(FINISH);
     move_abs_2(xpos,ypos);
     set_text_index(header_color);
     text("EXIT and SAVE Commands");
     move_rel_2(0.0,1f-0.5);
     set_text_index(hilite);
    set_pick_id(1);
     text("Exit - save changes.....ZZ");
     move_rel_2(0.0,1f);
    set_pick_id(2);
     text("Exit - no save.........:q!");
     move_rel_2(0.0,1f);
    set_pick_id(3);
     text("Save do not Exit.......:w");
     set_text_index(text_color);
     move_rel_2(0.0,1f);
    set_pick_id(4);
     text("Save lines n1 to n2....:n1,n2w");
     move_rel_2(0.0,1f);
```

```
    set_pick_id(5);
     text("Save to other file..:w filename");
     move_rel_2(0.0,1f);
    close_retained_segment();
    set_segment_detectability(FINISH,TRUE);
    set_segment_visibility(FINISH,FALSE);
   }

/*
        Exit the VI editor                                          */

Finish_msg1 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(FINISH_MSG1);
    move_abs_2(xpos+sp,ypos);
    text("Quitting the VI Editor:");
    move_rel_2(0.0,1f);
    text("Typing 'ZZ' will get you out of VI and save whatever file you");
    move_rel_2(0.0,1f);
    text("were working on.  If you want to get out without saving your");
    move_rel_2(0.0,1f);
    text("work type ':q!' .  That does an unconditional quit, abandoning");
    move_rel_2(0.0,1f);
    text("all changes to the file that weren't previously saved.  It can");
    move_rel_2(0.0,1f);
    text("be used to minimize damage if you've had a mental lapse don't");
    move_rel_2(0.0,1f);
    text("want to clobber your original file with nastiness.");
    close_retained_segment();
   } /*  End of Finish_msg1 */

/*
        Saving your work without exiting the editor
```

```
                                                                    */

Finish_msg2 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(FINISH_MSG2);
    move_abs_2(xpos+sp,ypos);
    text("Saving Changes:");
    move_rel_2(0.0,1f);
    text("The write command ':w' will save the file you are working on");
    move_rel_2(0.0,1f);
    text("without leaving the editor.  The full syntax of the command is:");
    move_rel_2(0.0,1f);
    text("  :(n1,n2)w(>>)(filename) .  Thus you may write specific lines to a");
    move_rel_2(0.0,1f);
    text("different file.  '>>' will cause the write operation to append");
    move_rel_2(0.0,1f);
    text("the new text to the existing file.  See Colon commands for more");
    move_rel_2(0.0,1f);
    text("info, also section 34.5 of the VI Tutorial, and section 8.3 of");
    move_rel_2(0.0,1f);
    text("An Introduction to Display Editing with VI.");
    close_retained_segment();
   } /*  End of Finish_msg2 */

/*
        A few miscellaneous notes are listed here.
                                                                    */

misc(xpos,ypos)  /* Miscellaneous notes about notation */
  float xpos,ypos;
  {
     short i;

    create_retained_segment(MISC);
```

```
        move_abs_2(xpos+sp,ypos);
        set_text_index(header_color);
        text("Miscellaneous Notes:");
        move_rel_2(0.0,1f);
        set_text_index(text_color);
        text("(n) - optional number or letter.");
        move_rel_2(0.0,1f);
        text("[a-z] - a range of values, pick only one.");
        move_rel_2(0.0,1f);
        text("<extent> - a delimiter for the operation: w,W,b,B,e,E,$,^,0,etc.");
        move_rel_2(0.0,1f);
        text("<esc> - escape key");
        move_rel_2(0.0,1f);
        text("<cr> - carriage return");
        move_rel_2(0.0,1f);
        text("^X - means simultaneous control key and x");
        move_rel_2(0.0,1f);
        text("Capital letters listed second have similar effect");
        move_rel_2(0.0,1f);
        text("This list is not exhaustive - see the VI tutorial on VAX Unix.");
        move_rel_2(0.0,1i);
        set_text_index(hilite);
        text("A minimal subset of commands is emphasized in this color.");
        set_text_index(text_color);
        move_rel_2(0.0,1f);
      close_retained_segment();
      set_segment_visibility(MISC,FALSE);
    }

/*
        Notes about buffers are listed here.                    */

buffers(xpos,ypos)
  float xpos,ypos;
```

```
{
    short i;

   create_retained_segment(BUFFERS);
    move_abs_2(xpos,ypos);
    set_text_index(header_color);
    text("Buffers:");
    move_rel_2(0.0,1f-0.5);
    set_text_index(text_color);
    set_pick_id(1);
    text("Save text to a buffer.....
    move_rel_2(0.0,1f);
    set_pick_id(2);
    text("Append to  text in buffer..
    move_rel_2(0.0,1f);
    set_pick_id(3);
    text("Get text from buffer...
    move_rel_2(0.0,1f);
    set_pick_id(4);
    text("Get text from buffer...
    move_rel_2(0.0,2*1f);
    text("Examples:");
    move_rel_2(0.0,1f);
    text("Two lines to buffer z.....
    move_rel_2(0.0,1f);
    text("End of line to buffer e.....
    move_rel_2(0.0,1f);
    text("");
    move_rel_2(0.0,1f);
   close_retained_segment();
   set_segment_detectability(BUFFERS,TRUE);
   set_segment_visibility(BUFFERS,FALSE);
}

/*
```

```
        Saving text into a buffer                                        */

Buffer_msg1 (xpos,ypos)
    float xpos,ypos;
    {
     create_retained_segment(BUFFER_MSG1);
     move_abs_2(xpos+sp,ypos);
     text("Saving to a Buffer:");
     move_rel_2(0.0,lf);
     text("The command
     move_rel_2(0.0,lf);
     text("   text to be stored in that named buffer, overwriting the previous");
     move_rel_2(0.0,lf);
     text("   contents.  The text may be from a character up to a whole file.");
     move_rel_2(0.0,lf);
     text("   The contents of the named buffer remain even though the file ");
     move_rel_2(0.0,lf);
     text("   being edited has changed.  Use named buffers when copying parts");
     move_rel_2(0.0,lf);
     text("   of one file into another.");
     move_rel_2(0.0,lf);
     text("Example:  Copy several lines of one file into another.");
     move_rel_2(0.0,lf);
     text("     Step 1.  type:  'vi filename1'");
     move_rel_2(0.0,lf);
     text("     Step 2.  position the cursor at the first line you want saved.");
     move_rel_2(0.0,lf);
     text("     Step 3.  type:  '
     move_rel_2(0.0,lf);
     text("     Step 4.  type:  ':e filename2'");
     move_rel_2(0.0,lf);
     text("     Step 5.  position the cursor where you want the new text.");
     move_rel_2(0.0,lf);
     text("     Step 6.  type:  '
```

```
    move_rel_2(0.2,1f);
    text("Large blocks af text may be copied very quickly this way.");
    close_retained_segment();
   } /*  End of Buffer_msg1 */


/*
        Saving text and appending to a buffer
                                                                    */

Buffer_msg2 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(BUFFER_MSG2);
    move_abs_2(xpos+sp,ypos);
    text("Appending to a Buffer:");
    move_rel_2(0.0,1f);
    text("Using a capital letter for the buffer name causes the text to be");
    move_rel_2(0.0,1f);
    text("  appended to the present contents of the buffer.");
    move_rel_2(0.0,2*1f);
    text("Numbered Buffers:");
    move_rel_2(0.0,1f);
    text("Numbered buffers are used to store the last 9 DELETIONS.  They");
    move_rel_2(0.0,1f);
    text("  are accessed the same way ");
    move_rel_2(0.0,1f);
    text("  of them type: ");
    move_rel_2(0.0,1f);
    text("Buffer 1 is the default buffer used by all commands that change");
    move_rel_2(0.0,1f);
    text("  or move text.  Buffers 2-9 are used only by the delete operator.");
    close_retained_segment();
   } /*  End of Buffer_msg2 */
```

```
/*
        Retrieving text from a buffer                                   */

Buffer_msg3 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(BUFFER_MSG3);
    move_abs_2(xpos+sp,ypos);
    text("Get text out of a buffer:");
    move_rel_2(0.0,lf);
    text("  Specify the name of the buffer and the put command.  The ");
    move_rel_2(0.0,lf);
    text("  buffer contents will then be displayed at the current cursor");
    move_rel_2(0.0,lf);
    text("  position.  You may retrieve text from both named and");
    move_rel_2(0.0,lf);
    text("  numbered buffers, but not simultaneously.");
    close_retained_segment();
   } /*  End of Buffer_msg3 */


/*
        Notes about moving text are listed here.                        */

move_text(xpos,ypos)  /* Steps for moving or copying text */
  float xpos,ypos;
  {
     short i;

    create_retained_segment(MOVE_TEXT);
     move_abs_2(xpos,ypos);
     set_text_index(header_color);
     text("Move or Duplicate Text:");
```

```
move_rel_2(0.0,1f-0.5);
set_text_index(text_color);
set_pick_id(1);
text(" Step 1.....Locate text");
move_rel_2(0.0,1f);
set_pick_id(2);
text(" Step 2.....Yank or Delete");
move_rel_2(0.0,1f);
set_pick_id(3);
text(" Step 3....Go to new location");
move_rel_2(0.0,1f);
set_pick_id(4);
text(" Step 4.....Put text");
move_rel_2(0.0,2*1f);
text("Copy Text Commands:");
move_rel_2(0.0,1f-0.5);
set_pick_id(5);
text("Yank syntax.....(n)y<extent>");
move_rel_2(0.0,1f);
set_text_index(hilite);
set_pick_id(6);
text("Yank line(s)....(n)Y or (n)yy");
set_text_index(text_color);
move_rel_2(0.0,1f);
set_pick_id(7);
text("Yank word(s).....(n)yw");
move_rel_2(0.0,1f);
set_pick_id(8);
text("Put text before cursor.....P");
move_rel_2(0.0,1f);
set_text_index(hilite);
set_pick_id(9);
text("Put text after cursor.....p");
set_text_index(text_color);
move_rel_2(0.0,1f);
```

```
    close_retained_segment();
    set_segment_detectability(MOVE_TEXT,TRUE);
    set_segment_visibility(MOVE_TEXT,FALSE);
   }


/*
        Moving and Copying text                                          */

Move_text_msg1 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(MOVE_TEXT_MSG1);
    move_abs_2(xpos+sp,ypos);
    text("Move or Copy Text:");
    move_rel_2(0.0,lf);
    text("The operations to move or copy text are identical:  one deletes");
    move_rel_2(0.0,lf);
    text("  the original copy, one does not.  The 'd' operator is used to");
    move_rel_2(0.0,lf);
    text("  move text; the 'y' operator to copy.  If no buffer is specified");
    move_rel_2(0.0,lf);
    text("  both use the default 'unnamed' buffer (#1 buffer).  Cursor");
    move_rel_2(0.0,lf);
    text("  position determines the starting location of the operation.");
    move_rel_2(0.0,lf);
    text("  The <extent> in the move or copy command specifies the end");
    move_rel_2(0.0,lf);
    text("  of the block.  Move the cursor to the new location, it also");
    move_rel_2(0.0,lf);
    text("  determines that position.  The put command 'p' or 'P' is used");
    move_rel_2(0.0,lf);
    text("  to put the text at the new position.");
    close_retained_segment();
```

```
} /*  End of Move_text_msg1 */

/*
        Moving and Copying text                                          */

Move_text_msg2 (xpos,ypos)
    float xpos,ypos;
    {
     create_retained_segment(MOVE_TEXT_MSG2);
     move_abs_2(xpos+sp,ypos);
     text("The Yank Command:");
     move_rel_2(0.0,1f);
     text("The command 'y' is used to make copies of chosen text.  It copies");
     move_rel_2(0.0,1f);
     text("  anything from a character upto an entire file, depending on the");
     move_rel_2(0.0,1f);
     text("  extent specified in the command.  'y' copies from the current");
     move_rel_2(0.0,1f);
     text("  cursor position.  'yy' or 'Y' copy from the current line.");
     move_rel_2(0.0,1f);
     text("  The optional number prefix means copy from here to the n-th");
     move_rel_2(0.0,1f);
     text("  occurrence of <extent>");
     move_rel_2(0.0,2*1f);
     text("Examples:");
     move_rel_2(0.0,1f);
     text("  Copy the next five words:........'5yw'");
     move_rel_2(0.0,1f);
     text("  Copy ten lines:........'10Y'..or..'10yy'");
     move_rel_2(0.0,1f);
     text("  Copy up the string 'curr':........'y/curr/'");
     move_rel_2(0.0,1f);
     text("");
```

```
    close_retained_segment();
   } /*  End of Move_text_msg2 */

/*
        Moving and Copying text                                          */


Move_text_msg3 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(MOVE_TEXT_MSG3);
    move_abs_2(xpos+sp,ypos);
    text("Putting text back:");
    move_rel_2(0.0,2*lf);
    text("The 'p' command puts text in the unnamed buffer to the right");
    move_rel_2(0.0,lf);
    text("  or below the current cursor position.  'P' puts it to the");
    move_rel_2(0.0,lf);
    text("  left or above the current position.");
    move_rel_2(0.0,2*lf);
    text("When used in conjunction with the named buffers the command is:");
    move_rel_2(0.0,lf);
    text("
    move_rel_2(0.0,lf);
    text("  in that buffer (
    close_retained_segment();
   } /*  End of Move_text_msg3 */

/*
        The text for the Colon commands is listed here.                  */


colon_cmds(xpos,ypos)  /* Colon command info */
  float xpos,ypos;
  {
```

```
 short i;

create_retained_segment(COLON_CMDS);
 move_abs_2(xpos,ypos);
 set_text_index(header_color);
 text("Colon Commands:");
 move_rel_2(0.0,lf);
 set_text_index(text_color);
 set_pick_id(1);
 text("Quit this file(no save)...:q!<cr>");
 move_rel_2(0.0,lf);
 set_pick_id(2);
 text("Save this file (write).....:w<cr>");
 move_rel_2(0.0,lf);
 set_pick_id(3);
 text("edit a file .....:e<filename><cr>");
 move_rel_2(0.0,lf);
 set_pick_id(4);
 text("Read a file in...:r<filename><cr>");
 move_rel_2(0.0,lf);
 set_pick_id(5);
 text("Go to line #n.............:n<cr>");
 move_rel_2(0.0,lf);
 set_pick_id(6);
 text("substitute text.....");
 move_rel_2(0.0,lf);
 text(".....:(n1,n2)s/t1/t2/(g)<cr>");
 move_rel_2(0.0,lf);
 set_pick_id(7);
 text("Execute a UNIX command..:!cmd<cr>");
 move_rel_2(0.0,lf);
 set_pick_id(8);
 text("Escape to a shell.....:sh<cr>");
 move_rel_2(0.0,lf);
 set_pick_id(9);
```

```
        text("Abbreviations.....:ab abr string");
        move_rel_2(0.0,1i);
        set_pick_id(10);
        text("Set options.....:se <option>");
        move_rel_2(0.0,1f);
      close_retained_segment();
      set_segment_detectability(COLON_CMDS,TRUE);
      set_segment_visibility(COLON_CMDS,FALSE);
    }


/*
        General info about colon commands                                    */

Colon_msg1 (xpos,ypos)
    float xpos,ypos;
    {
     create_retained_segment(COLON_MSG1);
     move_abs_2(xpos+sp,ypos);
     text("Colon Commands:");
     move_rel_2(0.0,2*1f);
     text("Colon commands are for the EX editor (another UNIX editor) upon");
     move_rel_2(0.0,1f);
     text("  which VI is overlayed.  These commands allow some operations");
     move_rel_2(0.0,1f);
     text("  not available in VI, and some duplication.");
     move_rel_2(0.0,1f);
     text("They are used primarily to read and write files into the editor, ");
     move_rel_2(0.0,1f);
     text("  to do global changes, and to escape to a 'C' shell.");
     move_rel_2(0.0,1f);
     text("  They allow you to do things around the editor without leaving");
     move_rel_2(0.0,1f);
     text("  it or losing your place in the file you are editing.");
```

```
        move_rel_2(0.0,1f);
        text("Example:    ");
        move_rel_2(0.0,1f);
        text("   While editing a program you want to see the effect of changes.");
        move_rel_2(0.0,1f);
        text("   Use ':w' to save the latest version, then ':sh' to escape to ");
        move_rel_2(0.0,1f);
        text("   a shell, where you compile and execute the new version.  When");
        move_rel_2(0.0,1f);
        text("   execution is complete and you have finished your analysis, type");
        move_rel_2(0.0,1f);
        text("   ^D' or 'exit' and you will be back in your file in VI");
        close_retained_segment();
      } /*  End of Colon_msg1 */


    /*
          Editing additional files from within the editor.
                                                                    */

    Colon_msg2 (xpos,ypos)
       float xpos,ypos;
       {
        create_retained_segment(COLON_MSG2);
        move_abs_2(xpos+sp,ypos);
        text("Edit Another File:");
        move_rel_2(0.0,2*1f);
        text("You may edit additional files in VI without quitting the editor");
        move_rel_2(0.0,1f);
        text("   Type ':e <filename> <cr>' to bring in the new file.  If you have");
        move_rel_2(0.0,1f);
        text("   not saved the changes to the previous file a warning will be");
        move_rel_2(0.0,1f);
        text("   written at the bottom of the screen.  To save the current file");
        move_rel_2(0.0,1f);
```

```
    text("  type ':w <cr>' then ':e <filename> <cr>.  If you want to");
    move_rel_2(0.0,1f);
    text("  abandon the changes anyway type ':e! <filename><cr>'.");
    move_rel_2(0.0,1f);
    text("If autowrite is set type ':n <filename><cr>' and your current");
    move_rel_2(0.0,1f);
    text("  file will be saved and the new one brought into the editor");
    move_rel_2(0.0,1f);
    text("  buffer.  See section 8.3 in An Introduction to Display Editing");
    move_rel_2(0.0,1f);
    text("  with VI.");
    close_retained_segment();
   } /*  End of Colon_msg2 */


/*
        Reading files into the VI editor buffer.
                                                                    */

Colon_msg3 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(COLON_MSG3);
    move_abs_2(xpos+sp,ypos);
    text("Reading Files:");
    move_rel_2(0.0,2*1f);
    text("The command ':r<filename>' will cause a copy of file <filename>");
    move_rel_2(0.0,1f);
    text("  to be read into the file you are working on in VI, at the ");
    move_rel_2(0.0,1f);
    text("  current cursor position.");
    move_rel_2(0.0,2*1f);
    text("To build modular programs make several files containing skeleton");
    move_rel_2(0.0,1f);
```

```
    text("  constructs, in a working directory.  Then building a program");
    move_rel_2(0.0,1f);
    text("  is just a series of ':r<filename>' commands editing each to");
    move_rel_2(0.0,1f);
    text("  'flesh out' the skeletons.  The program to do this screen was");
    move_rel_2(0.0,1f);
    text("  done largely this way.");
    close_retained_segment();
   } /*  End of Colon_msg3 */

/*
        Substitute text in a file automatically
                                                                    */

Colon_msg4 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(COLON_MSG4);
    move_abs_2(xpos+sp,ypos);
    text("Substitute Text:");
    move_rel_2(0.0,2*lf);
    text("The command to substitute text works just like the one in BTED");
    move_rel_2(0.0,1f);
    text("  in CP/M.  It is an ed editor command.");
    move_rel_2(0.0,1f);
    text("  (n1,n2) are optional line numbers for the substitution range.");
    move_rel_2(0.0,1f);
    text("  Without them it will occur only on the current line.");
    move_rel_2(0.0,1f);
    text("  't1' is the thing you want replaced, 't2' is the replacement.");
    move_rel_2(0.0,1f);
    text("  The optional(g) means replace everywhere on a line, ");
    move_rel_2(0.0,1f);
    text("  otherwise only the first occurrence will be replaced.");
    move_rel_2(0.0,2*lf);
```

```
    text("'Magic' characters also work in the substitute command.  Thus,");
    move_rel_2(0.0,1f);
    text("  /.ting/ would match and substitute for sting or String");
    move_rel_2(0.0,1f);
    text("  See ed(1) in the UNIX user's manual for more info on the 's'");
    move_rel_2(0.0,1f);
    text("  command.  See An Introduction to Display Editing with VI,");
    move_rel_2(0.0,1f);
    text("  section 8.4 for more info on magic characters.");
    close_retained_segment();
   } /*  End of Colon_msg4 */


/*
        Executing shell commands from within VI
                                                                    */

Colon_msg5 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(COLON_MSG5);
    move_abs_2(xpos+sp,ypos);
    text("Execute Shell Command(s):");
    move_rel_2(0.0,2*lf);
    text("':!<cmd>' allows you to execute a single shell command from within");
    move_rel_2(0.0,1f);
    text("  the editor.  At the completion of the command type <cr> to");
    move_rel_2(0.0,1f);
    text("  continue in the editor.  It's also possible to give another");
    move_rel_2(0.0,1f);
    text("  colon command before the <cr>.");
    move_rel_2(0.0,2*lf);
    text("':sh<cr>' invokes a new shell from within VI.  You are free to ");
    move_rel_2(0.0,1f);
    text("  execute any legal command sequence.  The editor is suspended ");
```

```
    move_rel_2(0.0,1f);
    text("  until you type ^D to exit the shell.");
    close_retained_segment();
   } /*  End of Colon_msg5 */


/*
        Using abtreviations                                              */


Colon_msg6 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(COLON_MSG6);
    move_abs_2(xpos+sp,ypos);
    text("Abbreviations:");
    move_rel_2(0.0,2*1f);
    text("The command :ab allows you to use short abbreviations for long");
    move_rel_2(0.0,1f);
    text("  strings.  At the start of your editing session enter all the");
    move_rel_2(0.0,1f);
    text("  abbreviations you want to use.  Then while in an input mode ");
    move_rel_2(0.0,1f);
    text("  just type the abbreviation and a space, and the string will");
    move_rel_2(0.0,1f);
    text("  appear in full.  abr is the short string, string is the thing");
    move_rel_2(0.0,1f);
    text("  you want to avoid having to type out.  Be careful of characters");
    move_rel_2(0.0,1f);
    text("  like <,>,/ etc. they have caused trouble in the past.");
    close_retained_segment();
   } /*  End of Colon_msg6 */


 /*
```

```
        Setable options in VI
                                                                    */

Colon_msg7 (xpos,ypos)
   float xpos,ypos;
   {
    create_retained_segment(COLON_MSG7);
    move_abs_2(xpos+sp,ypos);
    text("Setable Options:");
    move_rel_2(0.0,2*lf);
    text("There are several options in VI which may be set to the user's");
    move_rel_2(0.0,lf);
    text("  liking.  Some are convenient when writing programs, others");
    move_rel_2(0.0,lf);
    text("  are better when composing documents.  They are set using the");
    move_rel_2(0.0,lf);
    text("  :set command.  See An Introduction to Display Editing with VI");
    move_rel_2(0.0,lf);
    text("  section 6.2 for more info.  This author likes:");
    move_rel_2(0.0,lf);
    text("  ignorecase, magic, shiftwidth=4, for documents; and");
    move_rel_2(0.0,lf);
    text("  those plus: showmatch, number, and autoindent for programs.");
    close_retained_segment();
   } /*  End of Colon_msg7 */


/*
        A few miscellaneous notes are listed here.
                                                                    */

gen_info(xpos,ypos)  /* General info for using these screens */
  float xpos,ypos;
  {
     short i;
```

```
create_retained_segment(GEN_INFO);
 move_abs_2(xpos+sp,ypos);
 set_text_index(header_color);
 text("General Information:");
 move_rel_2(0.0,2*lf);
 set_text_index(text_color);
 text("These two pages of VI editor commands and memory helpers will");
 move_rel_2(0.0,lf);
 text("   allow you to use the editor while learning about it. The assump-");
 move_rel_2(0.0,lf);
 text("   tion is that you have used a micro-computer before.");
 move_rel_2(0.0,2*lf);
 text("The mouse and pad are used to move the `printer's fist`.");
 move_rel_2(0.0,lf);
 text("   Place the fist above the item you wish to pick and press the ");
 move_rel_2(0.0,lf);
 text("   left button.  If you get no response move the fist higher or ");
 move_rel_2(0.0,lf);
 text("   lower and try again.  The right button resets the mouse.");
 move_rel_2(0.0,2*lf);
 text("Single quote marks are used to set commands off from the text,");
 move_rel_2(0.0,lf);
 text("   don't type them in your commands.");
 move_rel_2(0.0,2*lf);
 text("Try any or all of the commands listed here.  If you get in trouble");
 move_rel_2(0.0,lf);
 text("   or get confused remember: 'u' (UNDO!) helps prevent insanity.");
 move_rel_2(0.0,2*lf);
 text("Picking 'Page 1' or 'Page 2' will display the entire page; each");
 move_rel_2(0.0,lf);
 text("   item listed may be picked separately.");
 move_rel_2(0.0,lf);
close_retained_segment();
}
```

```
/*
        The erase routine simply draws a filled rectangle over the message
  area, writting over any text that happens to be there.  Care must be taken
  to erase the text separately, because that segment will not write again,
  until it has been turned off (set_segment_visibility(XXXXXX, FALSE)).
                                                                        */


static float erasex[] = {0.0,94.0,0.0,-94.0,0.0};
static float erasey[] = {0.0,0.0,-22.0,0.0,22.0};
erase(xpos,ypos)
  float xpos,ypos;
  {
    create_retained_segment(ERASE);
    set_line_index(blue);
    set_fill_index(black);
    set_segment_visibility(ERASE,FALSE);
    move_abs_2(xpos,ypos);
    polygon_rel_2(erasex,erasey,5);
    polyline_rel_2(erasex,erasey,5);
    close_retained_segment();
    set_segment_detectability(ERASE,2);
  }

/*
        The first cursor related message - using the arrow keys for cursor
  positioning.                                                          */


VI_cursmsg1()
  {
    create_retained_segment(CURS_MSG1);
    move_abs_2(VI_messageX+sp,VI_messageY);
    text("Single Character Cursor Control:");
    move_rel_2(0.0,1f);
    text("    The arrow keys at the upper right of the keyboard, move the");
```

```
    move_rel_2(0.2,1f);
    text("cursor around the screen.  The up and down key will cause the page");
    move_rel_2(0.0,1f);
    text("to shift one line per depression when at the upper or lower edge.");
    move_rel_2(0.0,1f);
    text("The right and left motion is restricted to the line the cursor is");
    move_rel_2(0.0,1f);
    text("on.  The cursor will move as long as the key is held down.  Other");
    move_rel_2(0.0,1f);
    text("keys share the arrow key function: 'h' = left, 'l' = right,");
    move_rel_2(0.0,1f);
    text("'j' = down, 'k' = up.  Also '+' = down, '-' = up.");
    close_retained_segment();
  }

/*
        The second cursor control message - Gross control on this screen.
                                                                        */

VI_cursmsg2()
  {
    create_retained_segment(CURS_MSG2);
    move_abs_2(VI_messageX+5,VI_messageY);
    text("Large Scale Cursor Control:");
    move_rel_2(0.0,1f);
    text("    The Commands:  h, L, M, move the cursor to the top, bottom, or");
    move_rel_2(0.0,1f);
    text("middle of the screen, with a single two key depression.");
    move_rel_2(0.0,1f);
    text("Notice these are upper case.  VI commands and operators are");
    move_rel_2(0.0,1f);
    text("all case sensitive.  Be alert!");
    close_retained_segment();
  }
```

```
/*
        The third cursor control message - Ends of the line.
                                                                */

VI_cursmsg3()
  {
    create_retained_segment(CURS_MSG3);
    move_abs_2(VI_messageX+sp,VI_messageY);
    text("Ends of the Line:");
    move_rel_2(0.0,1f);
    text("    The commands 0 (zero), and $, move the cursor to the beginning");
    move_rel_2(0.0,1f);
    text("or end of the current line, respectively.  They are also used");
    move_rel_2(0.0,1f);
    text("generically in other commands to denote line beginning or end.");
    move_rel_2(0.0,1f);
    text("See section 25 in the VI tutorial or section 4.2 in the Intro.");
    close_retained_segment();
  }

/*
        The fourth cursor control message - move the cursor a word at a time.
                                                                */

VI_cursmsg4()
  {
    create_retained_segment(CURS_MSG4);
    move_abs_2(VI_messageX+sp,VI_messageY);
    text("Cursor Moves a Word at a Time:");
    move_rel_2(0.0,1f);
    text("    The cursor may be moved a word at a time using w, b, or e.");
    move_rel_2(0.0,1f);
    text("'w' moves the cursor forward a word, 'b'moves the cursor backward.");
    move_rel_2(0.0,1f);
    text("'e' moves the cursor forward stopping at the end of the word." );
```

```
    move_rel_2(0.0,1f);
    text("The lower case letters count punctuation as words, upper case ");
    move_rel_2(0.0,1f);
    text("ignores it. The command may be prefaced with a number to move ");
    move_rel_2(0.0,1f);
    text("the cursor that number of words in one command sequence.");
    move_rel_2(0.0,1f);
    text("See section 32 of the VI tutorial for more info. ");
    close_retained_segment();
}
/*  The end of vicomms.c  */
```

LIST OF REFERENCES

1. Apple Computers Inc., MACINTOSH sales brochure, 1983.

2. Smith, D. C.; Irby, C.; Kimball, R.; Verplank, B.; Harlsem, E.; "Designing the Star User Interface"; BYTE Magazine , April, 1982, pp 242-282.

3. Shneiderman, B.; "Human Factors Experiments in Designing Interactive Systems", Computer , December 1979, pp. 9-19.

4. Card, Stuart K.; Moran, Thomas P.; Newell, Allen; The Psychology of Human-Computer Interaction , Erlbaum, 1983.

5. MacLennan, B. J.; Principles of Programming Languages: Design, Evaluation, and Implementation , Holt, Rinehart and Winston, 1983, p. 527.

6. Teitelman, W.; Masinter, L.; "The Interlisp Programming Environment", Computer , 14:4, April 1981, pp. 25-34.

7. Dwyer, Barry; "A User Friendly Algorithm", Communications of the ACM , September, 1981, Vol. 24, No. 9, pp. 556-561.

8. Foley, James D.; Van Dam, A.; The Fundamentals of Interactive Computer Graphics , Addison-Wesley, 1982.

9. Heckel, P.; "How to Design User Oriented Software", The Elements Of Friendly Software , Symposium by Quickview Systems, Los Altos, CA, 1982.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3. Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943	2
4. Curriculum Officer, Code 37 Computer Technology Naval Postgraduate School Monterey, California 93943	1
5. Professor George A. Rahe, Code 52 Computer Science Department Naval Postgraduate School Monterey, California 93943	1
6. LCDR Frederick E. Groenert HSL-36 Naval Station Mayport, Florida 92228	2

END

FILMED

3-85

DTIC