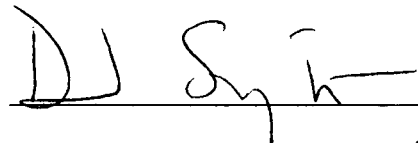


To the Graduate Council:

I am submitting herewith a thesis written by Regis S. Loffman entitled "User Interface for a Partially Incompatible System Software Environment with Non-ADP Users." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.


Charles P. Pfeleger, Major Professor

We have read this thesis
and recommend its acceptance:


Maria Zemanova

Accepted for the Council:


Vice Provost
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Regis J. Hoffman
Date March 2, 1987

USER INTERFACE FOR A PARTIALLY INCOMPATIBLE
SYSTEM SOFTWARE ENVIRONMENT
WITH NON-ADP USERS

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Regis S. Loffman

June 1987

ACKNOWLEDGMENTS

I am grateful to Dr. Charles Pfleeger, Dr. Maria Zemankova, and Dr. David Straight for their cooperation and assistance as members of my thesis committee. I am most appreciative of the guidance provided by Dr. Pfleeger in his role as head of the committee.

I would also like to acknowledge Martin Marietta Energy System's corporate support of my academic pursuits.

I extend a very sincere thanks to the management of Oak Ridge National Laboratory and especially to my colleagues for their personal and professional support over the past three years.

I am also thankful to my husband for his patience and support.

ABSTRACT

Good user interfaces to computer systems and software applications are the result of combining an analysis of user needs with knowledge of interface design principles and techniques. This thesis reports on an interface for an environment (a) that consists of users who are not computer science or data processing professionals and (b) which is bound by predetermined software and hardware. The interface was designed which combined these considerations with user interface design principles.

Current literature was investigated to establish a baseline of knowledge about user interface design. There are many techniques which can be used to implement a user interface, but all should have the same basic goal, which is to assist the user in the performance of a task. This can be accomplished by providing the user with consistent, well-structured interfaces which also provide flexibility to adapt to differences among users.

The interface produced used menu selection and command language techniques to make two different operating system environments appear similar. Additional included features helped to address the needs of different users. The original goal was also to make the

transition between the two systems transparent. This was not fully accomplished due to software and hardware limitations.

TABLE OF CONTENTS

CHAPTER	PAGE
I. INTRODUCTION	1
Background	1
Purpose	2
Approach	3
II. USER INTERFACE DESIGN CONSIDERATIONS	6
User Characteristics	8
User Interfaces	9
Different Dialogue Types	12
Question-and-Answer	12
Form Filling	13
Menu Selection	14
Function Keys	14
Query Language	15
Command Language	15
Graphic Interaction	16
Natural Language	16
Hybrid Dialogues	17
Parallel Dialogues	18
Dialogue Chosen for this Project	18
Menu Selection Design Considerations ...	19
Command Language Design Considerations..	22
Other Design Considerations	24

CHAPTER	PAGE
III. APPLICATION ENVIRONMENT AT HAND	28
Hardware-Software Configuration	29
User Needs	30
IV. IMPLEMENTATION DESCRIPTION	35
Implementation Plan	35
Analysis of Appropriate User Interaction .	37
Work Done for the First Kind of User	38
Command Implementation Details	42
Description of Individual Implemented	
Commands	43
Append	44
Copy	44
Create Directory and Create File	45
Delete	47
Edit	47
Files	48
Print	48
Remove Directory	49
Rename	54
Set Directory Protection and Set	
Protection	54
Sort	56
Type	57
Help Facility	58

CHAPTER	PAGE
IV. (Continued)	
Work Done for the Second Kind of User	58
Menu Characteristics	59
Return/Undo Capability	63
V. RESULTS, CONCLUSIONS, AND RECOMMENDATIONS ..	65
Desirable Changes	65
Handling Duplicate BTOS-Centix Names	67
The Ideal Implementation	70
BIBLIOGRAPHY	73
APPENDICES	78
APPENDIX A	79
APPENDIX B	84
APPENDIX C	88
APPENDIX D	104
APPENDIX E	106
APPENDIX F	108
APPENDIX G	114
APPENDIX H	187
APPENDIX I	195
VITA	210

CHAPTER I

INTRODUCTION

End user computing needs are becoming increasingly important as hardware and software proliferate in our society. This is evident in the variety of applications available, and the variety is expected to grow as users become more confident. It is tedious to the user, especially the user who is unfamiliar with computers and data processing, to cope with idiosyncracies of software. The issue is further complicated when the user has several software applications each with its own unique language or interface. The situation can be even worse when the hardware environment is not designed for the application.

This thesis presents the design and partial implementation of a user interface to integrate a system composed of multiple software applications in a partially incompatible system software and hardware environment.

Background

The particular environment being studied in this thesis is a government agency with over one hundred offices trying to make best use of a set of available tools. The user environment consists mostly of personnel

specialists who use the system for word processing and access to data base applications.

Not much is known about the capabilities and limitations of the total hardware-software environment. The hardware, the software which supports it, and any software applications developed will be distributed to approximately 200 sites within the United States and overseas. A better interface is needed for any application in the environment to be successful. The subject of this thesis is the development of a better interface.

Purpose

This thesis reports on a user interface for access to an integrated system. The user accesses a variety of software applications, each of which has a unique set of system software components. Because the typical user is unfamiliar with automated data processing (ADP), an important goal of the interface is that it minimize apparent differences between the various components. A way to accomplish this is to make access, use, and syntax of these components as consistent as possible. No attempt will be made in the interface effort to optimize execution or access time of the underlying software.

This thesis reports on user interface design considerations, the environment in which the integration is to

take place, the resulting interface that will integrate the components, and the overall ability and effectiveness of the attempt to integrate the components in a manner transparent to the user.

The focus of the thesis is not the details of making software run in the particular environment. The focus is the analysis of user needs and abilities and then the application of the principles of interface design to the design of a user interface which will make the different components more comparable.

The result of this thesis was a prototype system intended to show the viability of an integrated human interface that uses both predetermined hardware and software, and a mode of presentation familiar to the users. The system produced was a demonstration project, intended to elicit user feedback. It is expected that there will be substantial change to the basic system as a result of that feedback.

Approach

System design in general and user interface design in particular are implemented through a combination of hardware and software tools and techniques. The scope of this thesis is limited by predetermined software and hardware constraints. The hardware with which the

interface is to be implemented is required to be a monochrome monitor (green on black screen), a parallel printer and a serial printer for output, and a keyboard for input (a detailed description of the hardware environment appears in Chapter III). The system software is defined by two very different operating systems and a limited set of development software. (A more detailed description of the hardware and software environment appears in Chapter III.)

The user community is large in number and widely dispersed geographically. The typical user is not familiar with automated data processing and uses the system to accomplish office automation tasks. The level of available assistance at each office for computer related problems varies.

Because there is no option to acquire additional hardware or software for all of the sites, the only flexible part is the user interface which can be built on top of the hardware and system software. The thesis work focuses on the introduction of flexibility into this environment by the development of a user interface using only the available resources.

The approach taken by this thesis is four-fold. First, an investigation of the current literature was conducted to establish a baseline of knowledge about user interface theory and practice. The results are presented

in Chapter II. User interface is the term used throughout this thesis, but in the literature it is also referred to as ergonomics, human factors engineering*, human engineering**, and other similar terms in different sources.

Second, an analysis of users' abilities, needs, and skill levels was done. Third, based on knowledge of the application environment (user, hardware, and software as described in Chapter III), a set of features was chosen to be implemented. The features and the rationale for their selection are described in Chapter IV.

Fourth, the last part of the thesis work involved the actual implementation of the selected features. The results of this effort are reported in Chapter V.

Appendices A through F illustrate features of the hardware-software environment available to the user. These are the features supplied by the manufacturer. Appendices G through I contain the code written for this thesis as a response to the very different system supplied components.

*Human factors engineering or ergonomics - "The common objective of all human factors work is simply stated: to achieve, through appropriate design, functional effectiveness of whatever physical equipment or facilities people use." [Foley et al. 14]

**Human engineering - "refers to the interaction between the design of tools and the resulting ease of use by people" [Crawford 302]

CHAPTER II

USER INTERFACE DESIGN CONSIDERATIONS

Before advances in technology made computers smaller, relatively inexpensive, and less intimidating, computers and computer software were available to and used only by professional computer scientists and data processors. They were basically inaccessible to the lay person. Their cost was so high that only major organizations (government offices or businesses) could afford them. Even within these organizations computers and the applications they ran were controlled by and contained within a data processing department or computing organization.

Today, this scenario is obsolete. Because of personal computing, computers are in widespread use, and the users range from the completely inexperienced user to the sophisticated programmer. Given this wide range of experience within the user community, it is difficult for one computer system to adequately address the wide scope of needs. This is where the user interface comes into play. The user interface is what allows the user and the computer to communicate. "An ultimate goal of systems design is for the user interface to no longer be a barrier to the use of the capabilities by the people who

want to use them" [Hammer]. Foley et al. refer to interactive graphics but their thoughts easily apply to computer systems in general:

When a person uses an interactive graphics system to do real work, he wants the system to virtually disappear from his consciousness so that only his work and its ramifications have a claim on his energy. [Foley et al. 13]

The user interface is accomplished through a combination of hardware and software techniques. As mentioned in the introduction, hardware selections are not an option in this situation and, therefore, only software options are available for the problem solution.

All software designers should take into account the differences between humans, software and hardware, and the strengths and weaknesses of each.

The human's thought patterns are associative, integrative, and diffuse; the program's thought processes are direct, analytical, and specific. These differences are complementary and productive because the homunculus does well what the human cannot. [Crawford 302]

("Homunculus" is a term which Crawford uses to refer to an intelligent being within the innards of the computer.)

By keeping aware of these differences, the designer can produce a system that can perform the types of activities which the user does not do well. In situations where the user's participation is necessary, the presentation of information can be structured to minimize

the user's less creative efforts, allowing the majority of the user's efforts to be devoted to creative thought processes.

User Characteristics

The consensus of the sources examined is that the designer must know the intended user of the system. The success of system and user interface design affects user acceptance of the system, user productivity, training costs, etc. [Foley et al. 14]. There is a wide range of users with high expectations and the ideal interface would be tailored to each individual user's needs and abilities.

Foley et al. note the difference between users in the following manner:

A knowledgeable user requires a wider range of facilities and finer, more precise tools than a less knowledgeable user, before he will regard the system as efficient, accurate, or pleasurable. The experienced user can tolerate a much higher apparent "memory load" with many fewer prompting features. Indeed, evidence indicates that a system design that works well for the inexperienced user can be unproductively slow, crude, and displeasing to the experienced user. [Foley et al. 18]

Similarly, Shneiderman [226] notes that with "experience and maturity, users resent the computer's dominance and prefer to use the computer as a tool."

Branscomb and Thomas suggest a layered interface approach which reflects the differences among users [Branscomb and Thomas 227]. This layering can be accomplished using different dialogue techniques appropriate to the different types of users. (Dialogue techniques will be discussed later in this chapter.) The introduction of more complex commands as a user's knowledge increases is also a way to achieve layering.

Several authors [Branscomb and Thomas 228, Botteril 397] refer to the user's profile which would keep note of which interface is appropriate to the individual user. This profile could also maintain a record of the user's progress and adapt the interface to the user's current level of interaction.

User Interfaces

The tailoring of a system to a particular user is not always possible due to the limitations of available resources (programming, hardware limitations, etc.). However, much can be done to improve systems and their interfaces by incorporating some of the following concepts and techniques.

If the user accesses more than one system, data base, etc., it is beneficial if systems are designed with consistency as a common goal [Botteril 393]. Consistency

in this context refers to dialogue design, command vocabulary and structure, procedural flow through each system, error handling techniques, etc.

Command vocabulary achieves consistency at two levels. One level is the semantics level which refers to the meanings of a language. The other is the syntax level which refers to how the words in the language convey the semantic meaning.

By having systems and interfaces available that support consistency, the user experiences reduced learning time. If the user learns one system and then is presented with another which is consistent with the first, less training will be required and the second system will be learned in a shorter period of time. By keeping systems consistent, what the user has learned is reinforced and the retention time will be increased. The user is less likely to be presented with an unfamiliar situation which in turn will increase the user's self-confidence and confidence in the systems themselves.

The ideal system is one in which the user can never make a mistake. Of course this is wishful thinking but a goal of each system should be to minimize the likelihood of user error. This can be accomplished by careful dialogue structuring. The user should be provided with help facilities which provide more information when needed. The user should be provided with the capability

to withdraw from a situation before taking an irreversible action. A further capability would allow the user to undo an action that turns out to be not what was intended or wanted [Botteril 398, Branscomb and Thomas 230].

When the user inevitably makes a mistake, the system should provide helpful, non-antagonistic messages which provide the user with enough information to correct the situation. The not uncommon situation of the user's being presented with a cryptic message, often just identified by a message number, should be avoided. This abbreviated message requires the user to refer to another source (a manual) to interpret the message. The author's experience has often entailed searching for a manual that, once found, reveals that either the page explaining the error is missing or that the message is just repeated in hard copy format with no solution offered.

Having established the user's needs and the overall goal of assisting the user, the designer must choose the dialogue form. Shneiderman [226] notes that when designing for novice users "every attempt should be made to make the user at ease, without being patronizing or too obvious." This can be applied to how dialogues are structured and how messages are constructed.

Different Dialogue Types

There are many different dialogue types from which to choose when designing the interface. Martin lists twenty-three "techniques of conversation" employing a screen and a keyboard to perform alphanumeric displays and input [Martin 87]. Cole et al. [217] have a shorter list of dialogue categories:

question-and-answer	function keys
form filling	command language
query language	graphic interaction
menu selection	natural language
hybrid dialogues	parallel dialogues

These ten categories seem to represent the range of dialogue types. The following is a composite discussion of each from the literature and their respective merits. They are presented in an approximate order from "appropriate for use by inexperienced users" to "appropriate for experienced users" as determined by the author.

Question-and-Answer

Question-and-answer is an effective technique for use with inexperienced users. The user is presented with a set of predefined, system-initiated questions, one at a time. This technique has a limited usage scope and is

best used in an environment with well-defined interaction sequences.

This type of dialogue can be (machine and human) resource intensive. Computer time is required to generate the screen and interpret the response. This method also requires telecommunication resources to transport the screen and keyboard information between the terminal and computer.

The user's effort is devoted to the actual typing of the responses to each question. This involves the physical typing activity and the time required to proceed through the series of questions.

Question-and-answer is not an appropriate technique for experienced users and frequent users who are likely to be bored by the familiarity and tedium of a lengthy process. However, this method requires little training and is of benefit to the inexperienced or occasional user. The users need to be aware of the acceptable input formats or provided with helpful prompts to guide them through the process.

Form Filling

Form filling is similar to question-and-answer, but the user is presented with a screen-generated form which requires that values be supplied for each parameter. As in question-and-answer techniques, the users need to be

aware of acceptable input formats for their responses. The process can be shortened by allowing the user to take default values for parameters. As with question-and-answer, the experienced user may find this a boring process.

Menu Selection

Menu selection is also a system-initiated type of dialogue and is appropriate for use by inexperienced and occasional users. It is more appropriate for leading a user through the possible options for an action as opposed to leading the user through a series of data entries. Menu selection requires less keyboard typing by the user than question-and-answer or form filling because the user generally only needs to type in a letter or a number indicating the selection choice. The user generally cannot eliminate any steps required to perform an action. As with the other two methods, menu selection can be a (machine and human) resource intensive technique.

Function Keys

Function keys are of assistance to the user by enabling lengthy or frequently used input sequences to be abbreviated with the use of one key or a combination of a few keys. The use of function keys requires some

training. To be useful to the novice user the combination of keys should be simple and not lengthy. Function keys may be used in combination with other dialogue methods.

Query Language

Query languages are associated with data bases and data base management systems. There are several different types of query languages available with commercial data base management systems. These query languages can be implemented using menus, keywords, templates, query-by-example, command languages, form filling, and other methods. To produce meaningful, valid results, the user should have knowledge of the underlying data schema to properly structure the query.

Command Language

Command language dialogue is an effective technique for use by experienced users. It generally permits commands to be expressed in a short, concise, specialized syntax which can appear cryptic and meaningless to the uninitiated and inexperienced.

Because of the concise syntax associated with command language dialogue, this type of dialogue is not machine resource intensive. It also requires less manual effort on the part of the user to type out the command. Command

language dialogue does require training in the meaning and required syntax of the commands. It is not appropriate for inexperienced or occasional users, and even experienced users sometimes are in need of assistance.

Graphic Interaction

Interactive graphics as presented by Cole et al. is not a separate type of dialogue but is a tool for presenting information to the user. It is used with the other types of dialogues and can be geared to the level of the user. By including graphics as a type of dialogue, Cole et al. acknowledge that some information is best communicated graphically. Also, some people are better able to comprehend things in graphic form as opposed to written form.

Graphic interaction capabilities are becoming more common due to hardware and software improvements and the advantages of presenting information graphically.

Natural Language

Natural language dialogue refers to a dialogue where the user communicates with the computer similar to the way two people communicate. This type of interaction requires a large amount of machine resources to interpret the syntax, semantics, and ambiguities associated with context-dependent human communication.

Martin also discusses natural language [Martin 37] dialogue, but his usage refers to voice communication and is therefore not a dialogue type which is implemented using a keyboard and display. However, he notes that natural language applications have difficulty dealing with human syntax. The problem is not so difficult if there is a limited vocabulary and sentence constructs to deal with [Martin 39]. Even though Martin's use of natural language refers to voice communication, natural languages in general suffer from similar problems with syntax and are better handled with limited, well-defined vocabularies.

Natural language dialogues often support or are supported by artificial intelligence/expert systems and are the focus of many recent research efforts.

Hybrid Dialogues

Responding to the fact that every tool (in this case each dialogue type) is not suited to every task, the system designer may use a hybrid dialogue composed of two or more of the previously discussed dialogue types. Some dialogue types are more suitable for particular classes of activities than others. Hybrid dialogues combine dialogue types to provide better support for an activity. This combination allows greater flexibility for use but is more complex to design and implement.

Parallel Dialogues

Parallel dialogues offer the user a choice of dialogue method appropriate to the user's level of knowledge and individual comfort. These dialogues acknowledge the fact that user requirements change as the users gain experience in the use of a system. Parallel dialogues can provide the transition from basic functionality requiring little skill to more complex functions requiring more skill.

For example, a new user may require a lot of assistance to accomplish a task. In this situation, menu selection, form filling, and question-and-answer would be appropriate techniques to structure dialogues. As a user gains experience these methods may prove to be too time-intensive and alternative methods which can respond to this growth can help the user.

Dialogue Chosen for this Project

For reasons which will be discussed in Chapter IV, menu selection and command language were chosen as the types of dialogue to be implemented in the work of this thesis. Therefore, more detail is provided here about design considerations with respect to these two techniques.

Menu Selection Design Considerations

As stated previously, menus are well suited for use by occasional and inexperienced users. Menus require little manual (typing) effort on the part of the user. The user need not remember commands nor their syntax. Menus provide a structure by which an application can be approached and worked through. They also help the user get started with the task at hand [Botteril 416]. However, menus suffer from inflexibility which can be offset by parallelism.

Karhan et al. address similar needs with regard to instructions for public telephone use. In their work they cite the need for "consistent language, graphics, and placement of information" [Karhan et al. 1829].

When many menus are designed for an application or for a system of applications, it is helpful to the user if all of the menus have similar appearances. For example, many titles appear consistently in the same place, perhaps centered at the top of the menu. The choices appear consistently in the same place on the screen and choices may be grouped in several ways. Items can be grouped alphabetically, by frequency of use, or by some logical connection [Foley et al. 26]. Choices which are more irreversible (deleting data, exiting the application) can be set off from the other choices to

help ensure that the user does not misread the menu and make a selection with adverse results.

Menus should contain options to exit from the menu, to return to previous menus, or to view help screens [Shneiderman 238]. The words comprising the menus should be kept simple and short, and computer jargon should be avoided.

Menus are limited by how much information can physically be placed on one screen. Furnas et al. note the following:

When there are many objects, a menu system must use a successive search method that relies on some kind of hierarchical tree or other presentation of the relations among the objects. How to do this in a way that leads to correct user choices at each level, to good overall performance, and to acceptable convenience are unsolved issues. [Furnas et al. 1802]

Foley et al. [26] note the work of Snowberry et al. and their findings that selection time and accuracy improved when broader menus with fewer levels of selection were used. However, there were no suggested numbers of levels to be used as guides when constructing menus.

Shneiderman, in his discussion of short- and long-term memory [224], refers to the work of George Miller and the "magical number seven plus or minus two" as it relates to a person's processing capacity. Miller's work suggests that seven units is the limit for information perceived by any sensory organ [Shneiderman 224]. In a

later section of his book where he discusses menus and other types of dialogues, Shneiderman does not suggest a structure for menus with regard to the number of levels and the number of options available at each level. Perhaps the "magical number seven plus or minus two" can be applied to menu structuring as well.

A drawback to the use of hierarchical menus is the necessity of traversing through the hierarchy once a user has become familiar with the application. Foley et al. address this issue and suggest the following remedy:

Hierarchical menu structures almost demand an accompanying keyboard or function key technique for more experienced users. These techniques make selection especially easy if each node and leaf of the tree has an unambiguous name, allowing a user to directly enter a known command or phase name. The menu system provides a backup if the user's memory fails. An alternative is to require unambiguous names for each entry within an individual menu. Then the experienced user, seeking to avoid direct interaction with menus, can enter the complete path name to a leaf node. [Foley et al. 26]

The complete path name to a leaf node adds a command language or mnemonic capability to hierarchical menus.

Once a user has gained sufficient experience with an application, the menu hierarchy may be tedious to sequence through or the user may want to use advanced functions [Botteril 416]. This can be handled by providing this type of user with an alternative to menus.

Command Language Design Considerations

As discussed previously, command language dialogue is an effective technique for use with experienced users. With this type of dialogue, these users perform tasks quickly and are more satisfied because a greater sense of control is felt [Shneiderman 240]. However, because of the short, precise, specialized syntax associated with this type of dialogue, attention should be given to the design of these dialogues.

There appear to be two overall guiding principles with respect to the design of command language dialogues. With regard to the command names themselves, they should be "unique, easy to type, memorable, and natural" [Streeter et al. 1808]. Secondly, "command structures should match the problem domain and the sequence of user thought processes" [Shneiderman 255]. Attention to both of these will help the user to learn the commands initially and to remember them for later use.

There are several ways to develop command names. The first would be to use the complete word or phrase which describes the action to be taken. This produces the most meaningful command names and a place to start in determining an appropriate abbreviated version of the command. At this point in the development of command names there are several different methods from which to choose.

Truncation permits the user to type enough of the command to distinguish it from any other command. This scheme is adequate for short commands but for lengthy commands may require that a lot of the command be keyed before reaching the point of uniqueness.

Standard system abbreviations developed with careful planning can provide for both shortness and uniqueness. However, if they are developed without user input, the resulting commands may not be meaningful to the user, which will require more learning time and attention to use.

Another method, contraction, omits word internal letters [Streeter et al. 1810]. The deleted letters most often are vowels. This method provides for uniqueness but may still result in lengthy command names.

Acronyms are the result of taking the first letter of each word in the command. This provides for shortness of command names but may encounter problems with uniqueness.

The decision about which approach to take in designing command names is a difficult one. Streeter et al. provide this as a summary to their work:

...truncation appears to be the best single abbreviation scheme. Truncation also best captures people's natural abbreviations in all environments except two--monosyllabic words and multiple-word terms. In these cases, we recommend using vowel deletion for the former and acronym formation for the latter. If, on the other hand, one's task requires generating full

names, given abbreviations (decoding),
vowel deletion abbreviations are better
than other rule-based schemes.
[Streeter et al. 1825]

Streeter et al. do not offer a recommendation regarding the length of command names.

Botteril, in his discussion of the design rationale of the System/38 user interface, recommends three-character abbreviations for words and the concatenation of these words to produce system level commands. Two character abbreviations generally do not provide for uniqueness, and more than three results in names that are too long [Botteril 401]. The rule scheme he uses to produce abbreviations takes the first letter of the word and two consonants which are prominent in the pronunciation of the word and which also help make it unique from other words [Botteril 400-401].

Whichever method is chosen for developing command names, the structuring of the commands themselves should also follow similar syntax. An example of this is the command name followed by any argument(s) and the use of wildcards if permitted. Adhering to similar syntax patterns will assist the user in the use of commands.

Other Design Considerations

Irrespective of the type of dialogue technique chosen, there are several additional considerations in

designing dialogues: closure, completeness, familiarity, and flexibility.

Several authors cited the need for closure [Shneiderman 225, Crawford 316]. Shneiderman's use of "closure" refers to the need on the part of the user to complete a task. Crawford defines closure differently than Shneiderman, "The essence of closure is the narrowing of options, the elimination of possibilities, and the placement of rock-solid walls around the user."

Even though the two definitions are different, they both point to the importance of keeping in mind that the number of and difficulty of tasks with which a user can deal differs among users. It is important that the user be able to complete tasks. This completion of tasks lets the user gauge personal progress and provides a sense of security, especially for the novice user [Crawford 318].

An example of the lack of closure is implementing a dialogue using the question-and-answer technique where, at some point in the dialogue, the user doesn't know how to respond to a question. At this point the user will either attempt a guess or terminate the task. Guessing can result in unpredictable, wrong, and even disastrous results. Termination requires that the user reinitiate the operation. In either situation, the user can feel a wide range of emotions from frustration to anger to

distrust of the system. A well designed interface should avoid situations which evoke these emotions in users.

Achieving closure is aided by the careful structuring of dialogues, the screening of input values, and helpful guidance when errors do occur.

Crawford also addresses the need for completeness in a computer language. Although his thoughts are directed more toward programming languages, they also apply to applications in general which constitute another type of human-computer interaction or language. He states that the "language must completely express all the ideas that need to be communicated between the computer and the user but it need not express ideas internal to either thinker's thought process" [Crawford 314]. This points to the need that the chosen dialogue type be able to address all of the needs of the user which have to be communicated with the system.

Another aspect to be considered in the design process which has been touched on previously is that the interface should resemble that with which the user is familiar [Crawford 314]. This requires again that the interface be simple and direct. This will help the user in learning the application because it will not be completely foreign.

The use of color, intensity of color, sound, highlighting errors, graphics, icons, and other techniques

should also be considered when designing user interfaces. They can be used to direct the user's attention to significant functions or to highlight errors.

When designing a user interface it is therefore most important to keep the user in mind. This includes knowing what is needed to help the user accomplish a task as well as the different needs of individual users. The designer must also know the different forms that dialogues may take. By combining knowledge of the user's requirements with knowledge of dialogue techniques, the designer is then prepared to choose an interface method and begin the design effort.

CHAPTER III

APPLICATION ENVIRONMENT AT HAND

The user environment is a set of distributed government personnel offices which employs computers to do word processing and data base applications. In addition to handling general office tasks, the system will be used to maintain records for all of the personnel located at the site. The average user is a personnel clerk or technician and not a computer scientist or data processing professional.

There are approximately 200 offices located throughout the United States and overseas. The offices vary in size and in the level of in-house computer support which they can expect.

Most questions which cannot be handled at a particular office are forwarded to one central information center for resolution. This would result in a heavy burden on the one information center if all 200 offices had problems they could not resolve themselves. (There is also limited support available from the manufacturer in the form of a hotline.) In addition, few centralized staff members are devoted to solving problems. Their data processing experience is limited primarily to main-frame computers and applications, not micro computers.

They have only recent training in the operating system, the data base management system, and the hardware. Consequently, it is important that the software be reliable, consistent, and easy to use by non-ADP staff when distributed to the 200 offices.

Hardware-Software Configuration

Most of the sources reviewed advocated that hardware is selected only after the system requirements have been specified. This sequence of events often does not occur in the development of real systems. This was the case with the system under study in this thesis. The hardware and development software were acquired before the system (functional) requirements were specified. These resource limitations are not subject to change and, therefore, place constraints on what can be implemented.

The hardware configuration is composed of a Burroughs XE550 "megaframe" with Burroughs B 26 intelligent workstations clustered off of the XE550. The XE550 is composed of multiple processors assigned specific functions (applications processing, file processing, cluster processing, storage processing, and communications processing).

The XE550 primary operating system, BTOS, serves as a file processor. Centix, an enhanced version of Unix

System V, is provided through the applications processor. Centix is normally used with PT1500 terminals. The configuration for the environment under study does not have this type of terminal; rather it must be emulated by the B 26 workstation using Intercom 1500 emulation software. Centix is accessed by invoking the Intercom 1500 emulator on the BTOS file processor, which in turn connects through the hardware and software links to the Centix applications processor. Appendix A shows the BTOS logon screen, logon procedure, and how to invoke Intercom 1500.

Available software pertinent to this project are C, and Cobol compilers, and the Ingres data base management system running under Centix. Ingres is a relational data base management system (DBMS) from Relational Technology Inc.

User Needs

Current user applications on BTOS consist primarily of word processing and some data base management using a BTOS data base management system. In addition, users require the capability to do some simple system level commands such as accessing the current date and time and performing basic file management.

There are currently no applications running under Centix. A prototype system to evaluate candidates for job openings is being designed for Centix, and it will use Ingres as its data base management system because the data management complexity cannot be handled under BTOS. System administrators and data base administrators will need commands to do file management.

It is expected that the system will evolve to where word processing and other similar applications will be done at the BTOS level and data base applications will be developed using Ingres to run under Centix. Therefore, the users, who are not ADP professionals, will be faced with two totally different operating systems.

The BTOS commands available to the typical user employ the form filling dialogue technique described in Chapter II. Appendix B lists the BTOS commands available to the typical user. This list is displayed to the user when the HELP key is pressed. After this list is completely displayed, if the user again presses the HELP key, more information will be displayed about these commands. The listing produced when the user takes this action is in Appendix C.

The following is a general description of the use of the BTOS system. A typical command, for example "create file", is typed by the user at the COMMAND prompt. The user then presses the return key and is presented with a

form (Appendix D) which requires values to be supplied for the parameters. When the form is filled in, the user presses the GO key to execute the command.

There are only two ways to somewhat shorten this sequence. First, the user need only type in enough of the command name to identify it uniquely. (This could be considered a type of command language as described in Chapter II.) Second, if the command has default values and the user is willing to accept those values, the user need only enter the command and press the GO key.

The BTOS data base management system's query language can be characterized as being a command language dialogue. Ingres allows queries to be constructed either by using form filling or by formulating command language statements. The applications developed using Ingres have access to menu selection techniques.

The complexity of the hardware/software environment described above has a direct bearing on the design of the system. The hardware and software described previously in this chapter have already been purchased through a large government procurement. This has resulted in a constraint: no new hardware or development software can be purchased for distribution to the sites. Foley et al. recommend designing the system to meet the users' needs and then acquiring the hardware and software needed to implement the design [Foley et al. 17]. In the system

under study by this thesis, the application designers are to make do with what is in hand. This approach is contrary to recommended system design techniques but is typical of real world situations.

Another design concern not related to the user pertains to who will be responsible for maintaining, supporting, and distributing the user interface code. It is assumed that this person (or persons) has minimal knowledge of the hardware/software environment. Therefore, any code produced will have to be well documented. The system administrator currently at each site is often not familiar with data processing. This person has the responsibility to install new software at the local site, and this situation may affect the complexity of some of what is to be developed.

As opposed to the BTOS form filling dialogue, Centix uses a command language dialogue. The command language available with Centix is virtually indistinguishable from "standard" Unix commands in spelling and syntax as well as in method of operation. Thus the need to use applications under both BTOS and Centix not only violates the consistency requirement for a user interface, but also forces the user to learn two very complex operating systems to perform a small number of office automation functions. The purpose of the work described here is to apply the methodology of user interface design to the

development of a more consistent set of forms and commands to make the existence of two operating systems more transparent to the user.

This chapter has characterized the environment for which the user interface was developed. The hardware and software were already purchased and were not chosen for design characteristics needed for this project. The system consists of two very different operating systems (BTOS and Centix) and a data base management system. These system components do not adequately address the needs of the user community which is composed primarily of non-ADP users. Because there is no option to acquire additional hardware or software, it was determined that a user interface was needed to shield the user from the differences between the incompatible system components.

The next chapter describes the work that was done which applies user interface design techniques (the subject of Chapter II) to the environment just defined.

CHAPTER IV

IMPLEMENTATION DESCRIPTION

The requirement to use the existing hardware and software limited the options available for designing and implementing the user interface. The only design options are software techniques that can be accomplished with existing development software on the existing hardware. Chapter II briefly mentioned the use of color, sound, and graphics as techniques for emphasizing information. These are examples of features which are not feasible given the hardware and software constraints of this particular environment.

Implementation Plan

Under the existing environment, the user first logs on to BTOS. BTOS has a distinctive log on screen (Appendix A), command line prompt (COMMAND), and vocabulary of commands. To get to Centix, the user first has to run the PT Emulation software, which brings up the Centix logon screen. Because BTOS is what the user interacts with first and has been located at the sites longer than Centix, the original plan for this thesis was to:

a. try to make the logon procedure from BTOS to Centix more transparent

b. make the Centix command language environment look more like the BTOS form filling environment

c. design a prototype dialogue for an application to suggest a way applications under Centix can be accessed by users.

The first item, that of a transparent log on procedure between BTOS and Centix, was not able to be accomplished. This was due primarily to the PT 1500 emulation software required to run between BTOS and Centix. Currently upon exiting the application the user is returned to the Centix shell prompt. The user then has to logout of Centix by pressing the FINISH key. To return to BTOS, the user then has to press the CODE and FINISH keys. It was intended that the user be automatically returned to BTOS or logged out to the Centix log on screen. To implement either of these requires knowing the escape sequences for the FINISH key and the CODE-FINISH key sequence and where to send these sequences once they are known. Both of these questions are not addressed accurately in the system manuals and have been referred to the manufacturer.

The remaining two items therefore are the subject of this thesis and will be discussed in terms of the user

who is most likely to require their specific functionality.

Analysis of Appropriate User Interaction

As was discussed in Chapter II, there are approximately ten different types of dialogue which can be used to facilitate communication between the user and the computer. After much thought it was decided to design software to help two different kinds of users in the office under study.

The first type of user is the user who interacts directly with Centix. This user is more experienced in computing than the second type of user. For the user who accesses Centix, form filling techniques that look like BTOS were used to develop commands. This satisfies several goals of interface design, for example consistency and familiarity because form filling techniques most resemble what is available under BTOS. Command language techniques were also developed for the user who has progressed beyond the need for form filling. This introduces flexibility and adaptation to individual user needs.

The second type of user is the user who uses Centix to access applications that were developed using Ingres. Therefore, this type of user will never need to do Centix

level commands (file listings, file deletions, etc.). Instead, access only to a particular application is needed. A menu system was developed to assist this type of user (the actual menu system is explained later). Menu selection is a good type of dialogue for a novice user because the user does not need to know complex syntax. Some enhancements to the menus were added using command language techniques for the user who does not need all of the structure provided by menus.

Work Done for the First Kind of User

Most users of this system have had experience with BTOS. Therefore it is important to make the Centix environment look like BTOS. One obvious difference is the different prompts in the two systems. BTOS prompts the user with "COMMAND" and Centix prompts with "\$". To make the two environments seem similar, the PS1 variable (the PS1 variable overrides the default setting for the command-level prompt) was set to "COMMAND:" in the user's .profile file to override the system default "\$". Appendix E lists the .profile file for a typical user.

BTOS commands and Centix commands are very different. Appendix B contains a list of commands available to the typical user when in BTOS. These commands were reviewed, and a subset of commands was chosen which have functional

counterparts in Centix. The implementation of the selected commands in Centix is helpful to the user because, as was discussed in Chapter II, consistency between BTOS and Centix dialogue structure will be enforced. These commands and their Centix counterparts are listed in Table 1.

These BTOS commands were then implemented in Centix using shell scripts to make them appear and operate as close as possible to the BTOS environment. Shell scripts were chosen for the development work because they are easy to program and debug. Some simple C programs were used to perform cursor control and lend inverse video capability. These C programs were used because Curses (screen handling software which operates under Unix) is not available with the PT 1500 emulator. The programs used are accessed from the shell scripts as commands and are:

1. "inverse" which turns on inverse video
2. "mvdwn n" which moves the cursor down n number of lines
3. "mvup n" which moves the cursor up n number of lines
4. "normal" which turns on normal video
5. "rectangle m n" which establishes a rectangular area for the current screen attribute (such as "inverse") with the current cursor position as the upper left hand

Table 1. BTOS Commands and Similar Centix Commands

BTOS Command	Similar Centix Command
append	cat
copy	cp
create directory	mkdir
create file	
delete	rm
edit	edit or vi
files	ls
print	lpr
remove directory	rmdir
rename	mv
set directory protection	chmod
set protection	chmod
sort	sort
type	cat

corner and m lines down by n characters wide

6. "thisline n" which places the cursor at column n of the current line

All of the script files are trapped for the user pressing the delete key and abnormally terminating the shell script. To ensure that the screen is not left in an abnormal state, the delete key trap turns on normal video when a user presses the delete key. The trap could also be set to ignore the pressing of the delete key, but none of the commands as implemented would cause harm if abnormally terminated.

At the beginning of the effort it was also planned to implement three other frequently used BTOS commands. These are "logout", "path", and "set time". "Logout" could not be implemented because of the need to know the escape sequences for FINISH and CODE-FINISH as described earlier in this chapter. The BTOS "path" command sets a path to a specific drive (fixed drive or floppy drive) and to a specific directory. This could not be implemented using shell scripts because this requires the shell script to change the current directory (cd) of the parent shell. This is not permitted using shell scripts [Kochan and Wood 239]. The system clock is solely controlled through BTOS and, therefore, the "set time" command could not be implemented in Centix.

Command Implementation Details

For each of the above-mentioned BTOS commands Appendix F lists what the user sees when one of these commands is chosen. Appendix G contains the shell script code to implement each of them in Centix. Appendix H is a shell script that provides a help file capability for the Centix implementations of the BTOS commands.

Each command operates in two modes. The first mode is for the inexperienced user. If the user enters the command without supplying arguments, the script assumes the user wants to be prompted for parameter input as in BTOS. This was implemented using the form filling dialogue technique. The parameters which have meaning in Centix were implemented.

The second mode is for the experienced user and allows the entering of parameters on the same line as the command without prompting. The user only has to supply the command name and the file name(s) (or directory name for directory related commands). Default values are assumed for the other parameters with the exception of the "sort" command. This was implemented using the command language dialogue technique with the exception that command names are not shortened. Command names were not shortened in order to preserve the BTOS naming convention.

This two mode approach was chosen for several reasons. The form filling mode was chosen because this is consistent with the BTOS implementation. This provides users, who are accustomed to the BTOS approach, with a sense of continuity and security. The command language mode was chosen for the experienced user who may become bored with the form filling mode's slowness and required detail.

Five of the commands (edit, print, set, sort, and type) conflict with existing Centix commands of the same name. This means that to implement these five commands in this thesis would require the user specifying the full path name to where each shell script is located or include the path where the shell script is located in the PATH variable. "Edit", "print", and "sort" provide similar functionality, however, "set" and "type" do not. Chapter V will discuss the alternatives for each to determine which should be implemented.

Description of Individual Implemented Commands

A detailed description of each command and how it was implemented follows.

Append

The "append" command in BTOS prompts the user for the files to be appended and the file to which they will be appended. The resulting file can also be the printer which results in a printed output of the appended files and no new file is created. Optionally, the user can choose to confirm each append operation before it is performed. The default value assumes the user does not want to confirm each. If the destination file does not exist it is created, and if it does exist it is overwritten by the appended files.

The "append" shell script command in Centix using the form filling mode functions the same as BTOS except for the capability of sending the files to the printer.

The mode for the experienced user allows the user to specify any number of files and assumes the last file is the destination file. The files, as in the form filling mode, cannot be sent to the printer.

Copy

The "copy" command in BTOS prompts the user for the file to be copied and the file to which it will be copied. The user has two options. The first checks if the user wants to overwrite the destination file if it already exists. The other option allows the user to confirm the copy operation before it is performed. The

default value assumes the user does not want to confirm it.

The "copy" shell script command in Centix using the form filling mode functions the same as BTOS.

The mode for the experienced user copies the first file to the second file. It assumes overwriting is permitted and that the user does not want to confirm the operation.

Create Directory and Create File

The "create directory" command in BTOS prompts the user for the new directory name, which can also include a volume name ("volume" refers to the hard drive or floppy drive where the directory will be located). The user can optionally set the protection level for the files contained in the directory, specify the maximum number of files to be contained in the directory, and specify passwords for the directory and the volume.

The "create file" command in BTOS is similar to the create directory command. It prompts the user for the file name to be associated with the new file. The user can optionally set the protection level for the file's volume or directory, specify a password for the file, set the protection level for the file, specify an initial sector size, and choose to overwrite a pre-existing file.

The two BTOS commands were implemented in Centix using the same shell script, "create." The first argument to the shell script is compared to the words "file" and "directory" to determine which type of create is being performed. Once that is determined, the absence of an argument indicates that the user has chosen the form filling mode. Otherwise, the file or directory is created.

The "create directory" shell script command in Centix using the form filling mode just prompts the user for the directory name. The system protection level default is automatically chosen. The BTOS volume and directory passwords and number of files limitations do not have a comparable Centix implementation.

The "create file" shell script command in Centix using the form filling mode prompts the user for the file name and allows the user the option of overwriting an existing file. The other options available in BTOS (passwords, file protection level, and sector size) were not implemented because they are not available in Centix.

The mode for the experienced user for the "create directory" command expects a directory name as an argument to the command. If the directory already exists, a message to that effect is displayed; otherwise the directory is created. The directory protection level is the system default.

The experienced mode for the "create file" command expects a file name. If the file exists, the user is prompted to permit overwriting. Otherwise, a new file is created. The file protection level defaults to the system value.

Delete

The "delete" command in BTOS requires the user to fill a form with the file names to be deleted and optionally, the user can choose to confirm each deletion before it is executed. The default value assumes the user does not want to confirm each.

The "delete" shell script command in Centix using the form filling mode functions the same as BTOS.

The mode for the experienced user expects a list of file names as arguments to the command and assumes that the user does not want to confirm each deletion.

Edit

The "edit" command in BTOS prompts the user for the name of the file to be edited. If the user supplies a user name then more than one user can edit files in the directory.

The "edit" shell script command in Centix using the form filling mode asks only for the name of the file to be edited. Multiple user access to directories in Centix

is controlled by protection levels and, therefore, the user name parameter was not implemented.

The mode for the experienced user expects only the name of the file to be edited as an argument to the command.

With the "edit" command, the user will actually be invoking the vi editor.

Files

The "files" command in BTOS prompts the user for the names of the files to be listed. The user has two options. The first checks if the user wants details of each file to be displayed. The default is for no detail information. The other option allows the user to send the listing to the printer. The default value assumes the user wants the information to be displayed on the screen and not the printer.

The "files" shell script command in Centix using the form filling mode functions the same as BTOS.

The mode for the experienced user displays the list of file names with no details displayed on the screen.

Print

The "print" command in BTOS prompts the user for the file to be printed and then has a number of optional parameters. The user can specify the queue to handle the

printing. The user can specify the number of copies to be printed if more than one. After a file is printed it can be deleted automatically. The default is to not delete the file. The user can also confirm the printing of each file, the default being not to confirm.

There are other parameters which accommodate special forms, print wheels, and print modes, form alignment, a time when the printing will occur, a security mode which requires a password, and a priority to be applied to the actual printing schedule.

The "print" shell script command in Centix using the form filling mode implements the parameters which can be accommodated using the Centix "lpr" command. The user is prompted for queue name, number of copies, deletion after printing, and the confirmation of each print operation. The defaults are the same as those taken by BTOS. The other BTOS prompts discussed in the previous paragraph were not implemented.

The mode for the experienced user prints one copy of each of the listed files to the default queue and does not delete the file after printing nor does the user confirm each print operation.

Remove Directory

The "remove directory" command in BTOS prompts the user for the name of the directory to be deleted. If the

volume or directory has a password it must be supplied. The user has two options. The first checks if the user wants all of the files in the directory deleted. If the user does not respond with "yes" then if there are files in the directory the directory will not be removed. The user can also optionally choose to confirm each file deletion.

The mode for the experienced user was not difficult to implement. Because Centix has a hierarchical file structure, directories can be nested. For the "remove directory" command this implies that before a directory can be removed it must be empty of files which, by the definition of file in Centix, includes directories. For this reason, the command language mode was chosen to be implemented in a limited manner. The user supplies a directory name and the shell script checks that the argument is first of all a directory. If it is a directory, it is then checked to make sure that it is empty. If it is empty of files, it is removed. If it is not empty it is not deleted and a message is displayed to the user stating that the directory can not be removed because it contains files.

The form filling mode method was the difficult command to implement. The resulting implementation, although functional, does not include the breadth of scope originally planned. Because the nesting of files

in Centix allows for many possible situations the command seemed to be a good candidate for recursion. The BTOS convention of letting the user specify ahead of time whether all files should be deleted or not and the ability to confirm each deletion before it is processed seemed to provide adequate safety measures against indiscriminate file and directory deletion which recursion could introduce.

Code development was based on this premise and the assumption that a recursive shell script could be implemented. In simple cases it worked. However, as more complicated cases were tested, the recursion did not work. It seemed to be able to push down through directory levels but popping back out clearly was not functioning correctly. The shell script seemed to be having difficulty dealing with hierarchical structures that became too deep (three levels) and too wide.

No discussion of recursion applied to shell scripts was found in the available reference material including the system's Centix manuals. At this point a simple recursive shell script was written to determine if there is a limit on the number of times a shell script can be called recursively. The script basically continued to call itself until a counter reached a predefined limit (the value of the counter was printed each time) and then the value in the counter was printed as the levels popped

back out. This shell script worked correctly with small values. When the limit was set to twenty-four the shell script broke down. The twenty-four levels could be pushed but at the point where the popping would occur the execution was hung and had to be aborted.

A few times when this occurred a message came to the screen but never remained long enough to be completely recorded. It was something to the effect that the number of fork processes had been exceeded. The execution of a shell script causes the spawning of a new process. Apparently there is some limit to the number of processes which can be running. Whether the recursion encountered a process limit for an individual user or a system limit is not clear. The problem was referred to the manufacturer.

At this point the pursuit of a recursive solution to the "remove directory" command did not seem to be profitable. There were three other courses of action from which to choose. The first would be to write a C program to implement the remove directory command using recursion. The second would be to use a variation of the "ls" command (with the -R option) to build a file containing the names of all of the files (and directories) in a directory and use that file as the base from which to do deletions. The third alternative would be to limit the functionality of the command.

The first alternative, that of implementing the command with a C program, was discarded because the premise of the thesis is to demonstrate user interface techniques (using shell scripts because they are easy to program and debug). The second alternative was not chosen because in other than simple cases it would seem to require a lot of processing resulting in very slow execution time.

The third alternative was chosen to implement the prompt mode as a compromise between functionality and execution time. The user is prompted for a directory name and the user can optionally choose to delete all the files in the directory and to confirm each deletion. If the user does not want to delete all files, the directory is checked to see if it is empty. If there are files in the directory, a message is displayed to the effect that there are files present and the directory cannot be removed.

If the user wants to delete all files, the directory is checked for the presence of subdirectories. If there are subdirectories present, a message is displayed saying that the directory can not be removed because it contains subdirectories.

Assuming there are no subdirectories, if the user chooses to confirm each file deletion, the user is then prompted to confirm the deleting of each file. If all

files are subsequently confirmed for deletion, the directory is then removed. If some files are not confirmed for deletion, they are not deleted, and the directory is not removed.

Rename

The "rename" command in BTOS prompts the user for the file to be renamed and the new name for the file. Optionally, the user can specify whether overwriting is acceptable if the new file name is the name of an existing file. The default is to not overwrite. The user can choose to confirm the rename operation before it is performed. The default value assumes the user does not want to confirm each.

The "rename" shell script command in Centix using the form filling mode functions the same as BTOS.

The mode for the experienced user expects the user to supply the old file name and the new file name as arguments to the command. If the new file name is the name of an existing file, a message to that effect is displayed to the user and the file is not renamed.

Set Directory Protection and Set Protection

The "set directory" protection command in BTOS prompts the user for the directory name, which can also include a volume name (volume refers to the hard drive or

floppy drive where the directory will be located) and the new protection level. The user can optionally specify a volume or directory password and can also confirm the protection level change before it is finalized.

The "set protection" command in BTOS is similar to the "set directory protection" command, except that it applies to file protection. It prompts the user for a list of file names and a protection level to be assigned to each file. The user can optionally specify a password for the file and can also confirm the protection level change before it occurs.

The two commands were implemented in Centix using the same shell script, "set." The first argument to the shell script is compared to "directory" and "protection" to determine which type of create is being performed. If the first argument is directory, the second is checked for being "protection." Once it is determined whether the user is concerned with file or directory protection, the absence of other arguments indicates that the user has chosen the prompt mode. Otherwise, the new file or directory protection is assigned.

The "set directory protection" shell script command in Centix using the form filling mode prompts the user for the directory name, the new protection level, and, optionally to confirm the change.

The "set protection" shell script command in Centix using the form filling mode is analogous to the set directory protection command. It prompts the user for the file name(s), the new protection level, and an optional confirmation.

The mode for the experienced user for the "set directory protection" command expects a protection level and a list of directory names to which the protection level will be assigned as arguments to the command.

The experienced mode for the "set protection" command for files works like the "set directory protection" except that it expects a list of file names instead of directories as arguments to the command.

Sort

The "sort" command in BTOS prompts the user for the name of the files to be sorted and the file in which the sorted files are placed. The file organization must either RSAM (Record Sequential Access Method), DAM (Direct Access Method), or ISAM (Indexed Sequential Access Method). The user also specifies the keys on which to perform the sort. These keys are embedded in the data. The user then can supply optional parameters which pertain to stable sorts, work files, log file, and user confirmation for malformed input records.

Centix has a "sort" command which was used in the implementation of this command. However, it is used to sort lines of files, and the "key" is a portion of the line specified by a beginning and ending position. To fully implement the BTOS sort command would require writing a utility to handle RSAM, DAM, or ISAM records.

The "sort" command implemented here was done to put a BTOS-like front-end on the Centix "sort".

The "sort shell" script command in Centix using the form filling mode asks for the name of the files to be sorted, the output file, and the part of the line on which to sort as specified by a beginning and ending position in the line.

The mode for the experienced user expects a beginning and ending position on which to sort and a list of files as arguments to the command. It is assumed that the last file in the list is the output file to which the sorted lines will be written.

Type

The "type" command in BTOS prompts the user for a list of files to be displayed on the screen. The user can optionally choose to confirm each before it is displayed to the screen.

The "type" shell script command in Centix using the form filling mode works the same as in BTOS.

The mode for the experienced user for the "type" command expects a list of file names to be displayed to the screen as arguments to the command.

Help Facility

A help facility for the implemented commands and their Centix counterparts was also implemented using a shell script. The command to access the facility is "unixhelp." If no argument is supplied, a screen is displayed with each command listed and synonyms for the command. The user then selects a command for which more detail is supplied. The detail contains information about the use of the command, the form of the command and its expected arguments, and in some cases, cautions about using the command (for example, possible overwriting).

If an argument is supplied, the argument is checked to determine if it is one of the commands for which help is available. If so, the detail that was explained in the previous paragraph is displayed. If help information is not available, a message to that effect is displayed.

Work Done for the Second Kind of User

A menu system was developed for the Ingres applications. (See Appendix I for the code; note that the menus contain calls to Ingres and the Ingres called code is not

included.) The menu system approach was chosen for two reasons. First, non-ADP users are more accustomed to menu systems. Second, menus are an effective tool for novice users because menus require little effort on the part of the user. Menus lead a user through an application and offer a sense of security to users who are new to unfamiliar with the application or computers in general.

The strict hierarchical nature of the menu system was modified to accommodate the frequent user who may find the use of menus tedious. This was accomplished by putting a unique alphanumeric label in the upper right hand corner of each menu. This label identifies the menu screen and, by specifying the label, the user can directly access the specified menu.

Menu Characteristics

Main menus have a character label which is an acronym representing the function of the module. For example, ACP is the label associated with the Personnel Office Main Menu screen, which is the first screen the user sees and from which all applications are accessed through the menu tree (ACP has meaning to the user). CE is the label on the Candidate Evaluation System Main Menu. It is listed on ACP as an available application. Currently there are no other applications developed and available,

however, new ones would be included as choices on this menu.

The CE main menu contains the modules which comprise the Candidate Evaluation System. Currently, KSA Item Bank (KSA) and Crediting Plan Bank (CP) are the only available modules. The KSA Item Bank Main Menu choices include KSA010, KSA020, KSA030, and KSARPT. The numbered identifiers were selected as a way to distinguish between modules. It is intended that they be replaced with labels which are meaningful to the user. The users have been asked to determine what these labels should be. This will provide added meaning to the user and make the labels less cryptic.

Report menus are labeled with the code which identifies the module (KSA) and "RPT" for report (KSARPT). Help menus are labeled with the code which identifies the module (ACP) and "HLP" for help (ACPHLP).

A typical session will first log the user into the ACP screen. From this screen the user can log out or choose an application, for example Candidate Evaluation. From the application, the user can be guided through the system by the menus or direct access to a menu can be achieved by typing its unique screen label.

If the user chooses to use the menus, all selections just require the typing of the number of the selection and pressing return. The only screens which require more

typing effort are the screens which are called by Ingres. These are the data entry, data modification, and data querying screens which were all developed using Ingres utilities and which are only called by the shell script code (Appendix I) developed for this thesis.

The principle of consistency was ensured by having all of the menus follow the same format. The unique code is listed in the upper right hand corner. The title of the menu is listed in upper case letters and centered on the next line. The choices available to the user are listed below the title. The first choice is always 0 which returns the user to the menu from which the current screen was called. Notice that this choice is separated from the other choices below it by one blank line. This is done to distinguish it from the other choices. The other choices are then listed.

One screen may contain a variety of selections. The number of selections for the menus in this thesis ranges from three to five. Similar menu selections are grouped together and separated from different grouping by a blank line. For example, the KSA ITEM BANK MAIN MENU choices 1 and 2 are grouped together because they both call modules which add, edit, delete, or retrieve data. Choice 3 appears by itself because it calls the reporting module menu. Choice 4 also appears by itself because it has a different function from the other choices.

All menus are contained within a box of asterisks (*). Below the box, the user is prompted for a selection and also reminded to press the enter key. The separation of these areas by the box is done to emphasize the difference between choices available to the user and the place where the user inputs a selection.

At any time that the user makes an invalid menu selection, an error message is displayed which repeats the user's choice and that it is in error. The display of what the user entered is done to help the user correct keyboard entry errors. Then the menu is repainted, and the user is prompted for another choice.

All report menus behave the same. This reinforces the consistency principle by presenting the user with information in a familiar format. The user is presented with a menu of the different reports which are available at that point in the menu process. The user selects a report choice, and then is prompted for how the report is to be printed: either to the terminal, in draft copy, or in final copy. The choice of terminal will display the report on the screen. Draft copy will print the report on the parallel printer which is connected to the XE550. Final copy will print the report on the serial printer which is connected to the XE550. The user must know where these two printers are located so that printed output can be picked up. It would be helpful if the user

were also given a message as to where the printers are located (i.e., what room) but because this software will be distributed to approximately 200 different locations, this feature is not feasible to implement.

Return/Undo Capability

Interface design principles suggest that the user be given an undo capability. Within this menu system, this feature is interpreted as allowing the user to return to the calling menu (the previous menu). This is accomplished by the last choice which is "r". This choice is deliberately not a number because choosing it does not call a specific module menu but returns the user to the calling screen. This is included in case the user forgets what was done previously. There is no undo capability, but at least the user can see the previous screen.

It was originally intended for the r capability to allow the user to flip back and forth between two menus. This would eliminate the need for the user to type menu labels repeatedly. However, when a shell function executes another shell function by calling its name, a pointer to the parent function is placed on a stack. When a return is executed from the called shell function, the return stack is popped and returns the pointer to the parent shell function for it to be executed. There is no

way to put a pointer to the child on the stack as the parent is being popped.

Because the calls to Ingres are slow and are beyond the control of the menu system itself, the user is presented with a message ("This will take some time...") which acknowledges that processing is going on. This is done to assure the user that an error has not been made. After this message appears on the screen, the Ingres utilities are running and each of them prints its own messages to the screen so that at least the user knows something is going on. Once the user has completed an activity which involves Ingres (adding, updating, deleting, retrieving data or running a report), the menu from where the selection was made is redisplayed. This is an application of Shneiderman's interpretation of closure. The user is assured that the task requiring Ingres has been completed because the menu displayed is the one from which the task was initiated.

CHAPTER V

RESULTS, CONCLUSIONS, AND RECOMMENDATIONS

Ideally the user should be unaware of changing from BTOS to Centix and returning to BTOS. As was mentioned in the previous chapter, it was not possible to accomplish this goal. To do so, more help would be needed from the manufacturer's support staff. It appears that major system software redesign may be required to present the multiple operating systems as a single entity.

Desirable Changes

The menu system developed for the personnel system will need to be refined with the feedback of users to ensure that it provides the functionality they require and captures the sequence of their activities correctly. Shell scripts proved to be a good prototyping tool for menus because they can be modified easily and do not require recompiling.

Once the functionality and sequencing of menus has been agreed upon with the user, the menu system should be converted to C to improve response time. The ability to flip back and forth between two screens as was attempted with the "r" feature could probably be better implemented

using global variables which save a reference to the called and calling menus. (These recommendations have been successfully implemented as part of the work of the ongoing project of which this thesis is a component.)

The shell scripts, which were written to implement BTOS commands in Centix, should be converted to C for faster execution. Shell scripts were chosen as a quick way to implement the selected commands with the knowledge that they would execute slowly. Using C and a system call to perform a Centix "cd", the "path" command can probably be implemented.

In order to make the commands accessible by all users, the shell script files should be located in a specific directory. The path to that directory is then included either in each individual user's PATH (in the .profile file) or in the system default PATH if it is determined that the majority of users will use the commands. This directory path should be chosen carefully since it is affected by other considerations, such as the commands that duplicate existing Centix system command names.

The commands that duplicate existing Centix system command names can be used by either making an adjustment to the user's PATH or providing the full specification of the path to where the command is located. The adjustment

to the user's PATH is not advised, and requiring the user to supply a full path name to a command is tedious.

Currently the operating system provides no system help files to assist the user and documentation is poor. The help feature developed for the BTOS commands implemented in Centix is a good place to begin addressing the users' requirements for system level help facilities.

Handling Duplicate BTOS-Centix Names

It is not recommended that the "edit" command be implemented in a real environment for several reasons. In addition to the problem of where to locate the command, the implementation only provides for the prompting of the file name and still requires that the user be familiar with the text editor. Most users who are experienced enough to be editing files probably are not in need of this simple form filling mode implementation.

The implementation of the "print" command described in this thesis needs some consideration before it is installed in the user's environment. The functions this version provides (i.e., those of specifying print queue, number of copies, the deletion of files after printing) can be and were accomplished using Centix commands. However, the ability to provide the inexperienced user

with a command that has a form similar to the more familiar BTOS environment has merit. The "print" command presented in this thesis uses the Centix "lpr" command. If it is determined that the users, in particular those to whom these new commands are available, will not need the system's "print" command, then it would be reasonable to implement a PATH change to gain access to this "new" "print" command.

The "remove directory" command as implemented here is adequate in the sense that a user cannot remove a directory that contains files without choosing the "delete all files" option provided by the form filling method. However, because BTOS does not permit nested subdirectories and Centix does, there is an inconsistency between the BTOS command and the command implemented here. The author recommends that the writing of a C program be investigated to implement the command.

Centix has a "set" command which is used to display currently set shell variables. Because Centix commands all consist of one "word" (or combination of characters not including white spaces), "set" or "set directory protection" or "set protection" will invoke the system's "set" command.

The functions this "set" command provides can be and were accomplished using Centix commands (chmod). The use of "chmod" requires an understanding of the three types

of Centix users, how to determine appropriate access privileges, and then how to assign them. However, the ability to provide the inexperienced user with a command that has a form similar to the more familiar BTOS environment has merit. The commands to set protection presented here provide consistency.

The use of the Centix "set" command may not be required by most of the users in the environment described in this thesis, but the ability to set protection levels is. If the users, in particular those to whom these new commands are available, will not need the system's set command, then it would be reasonable to implement a PATH change to gain access to this different "set" command.

It is not recommended that the "sort" command developed in this thesis be implemented in a real environment without further discussion with the users for several reasons. There is a major difference in the intended use of the BTOS and the Centix sorts. The BTOS "sort" is used on records with a special structure. The Centix "sort" is used to sort lines of files. To implement the Centix "sort" with a BTOS front-end may be deceptive and confusing.

Centix has a "type" command which gives the path to the argument which is a Centix command. The need for the Centix's "type" command should be examined and

weighed against the need for a BTOS-like "type" command. If it is determined that the Centix system's command is not needed by the users and that the BTOS function is needed, then the choice to change the user's PATH would be reasonable.

The Ideal Implementation

The work developed in this thesis was done based on knowledge of the intended user community but without their direct input. Designing user interfaces requires knowledge of the user and user input. However, sometimes the users either don't know what they want or what is possible to implement given the resources (hardware and software) at hand. Consequently, the ideal situation is one in which the development of interfaces is a joint effort between the designer and user with the process being iterative. The work presented here has applied user interface theory to a real situation. The resulting menus and commands are one step in the iterative process to produce a user interface. That process will continue because the work presented in this thesis is part of an ongoing project which provides support to the particular government agency.

The goal of the work presented in this thesis was to apply current user interface design techniques to a

specific environment. The environment is one which consists of users with little experience with computers confronted with a hardware environment that has two very different operating systems.

Research into current user interface design practices and methodologies was conducted. The information learned from this exercise was then combined with an analysis of user needs and abilities to develop a user interface. The development of the interface had an overall goal of reducing the visible differences between two very different systems by making the interfaces consistent and familiar.

This goal was achieved by creating a set of commands (using form filling) for the one operating system (Centix) that resembles the same set of commands in the other operating system (BTOS). The commands were enhanced to accommodate differences in individual users by the addition of a command language syntax.

In recognition of the fact that one type of user will use Centix only to access applications developed with a data base management system, a menu system for an application was designed. The menu system design was based on menu system design guidelines. Features were added to respond to frequent users' need to quickly access the menu system without traversing the complete menu hierarchy.

If there were an unlimited supply of resources (human, software, and hardware) to develop a user interface for the environment described in this thesis, the approach to that development would have been different. The work would have started with an analysis of the needs of the users. Users would be involved in this process.

Once the requirements had been documented, an analysis of how to meet these requirements would be conducted. The first consideration would be the types of software needed and the last consideration would be the hardware. The possible choices and combinations are many and could be the topic of another thesis.

This thesis has demonstrated how sound user interface design principles can be successfully applied even to a situation which is bounded by many constraints.

BIBLIOGRAPHY

BIBLIOGRAPHY

Botterill, J. H., "The Design Rationale of the System/38 User Interface," IBM Systems Journal, Vol. 21, No. 4, 1982, pp. 384-423.

Branscomb, L. M., and J. C. Thomas, "Ease of Use: A System Design Challenge," IBM Systems Journal, Vol. 23, No. 3, 1984, pp. 224-235.

Burroughs Corporation, B20 Systems Standard Software Operations Guide, Burroughs Corporation, Detroit, Michigan, 1985.

Burroughs Corporation, XE 500 Centix System User's Guide, Volume I, Burroughs Corporation, Detroit, Michigan, 1986.

Butler, T. W., "Computer Response Time and User Performance During Data Entry," The Bell System Technical Journal, July/August 1984, Vol. 63, No. 6, Part 2, pp. 1007-1018.

Christie, Bruce (editor), Human Factors of Information Technology in the Office, John Wiley and Sons, Chichester, 1985.

Coke, E. U., and M. E. Koether, "A Study of the Match Between the Stylistic Difficulty of Technical Documents and the Reading Skills of Technical Personnel," The Bell System Technical Journal, Vol. 62, No. 6, Part 3, July/August 1983, pp. 1849-1864.

Cole, Ian, Mark Lansdale, and Bruce Christie, "Dialogue Design Guidelines (Chapter 10)," Human Factors of Information Technology in the Office (ed. Bruce Christie), John Wiley and Sons, Chichester, 1985, pp. 212-241.

Crawford, Chris, "The Atari Tutorial - Part 10: Human Engineering," BYTE, June 1982, pp. 302-318.

DATAPRO Research Corporation, "Burroughs B 25," DATAPRO Reports on Microcomputers, DATAPRO Research Corporation, Delran, New Jersey, July 1985, pp. CM11-117MM-101 - CM11-117MM-107.

DATAPRO Research Corporation, "Burroughs XE500 Series," DATAPRO Reports on Micros and Personal Computers, DATAPRO Research Corporation, Delran, New Jersey, July 1985, pp. M09-112-301 - M09-112-308.

Dzida, W., S. Herda, and W. D. Itzfeldt, "User-Perceived Quality of Interactive Systems," IEEE Transactions on Software Engineering, July 1978, Vol. SE-4, No. 4, pp. 270-276.

Field, Anne R., "The Next Boom in Computers: Services," Business Week, July 7, 1986, No. 2954, pp. 72-73.

Foley, James D., Victor L. Wallace, and Peggy Chan, "The Human Factors of Computer Graphics Interaction Techniques," IEEE Computer Graphics and Applications, November 1984, Vol. 4, No. 11, pp. 13-48.

Furnas, G. W., T. K. Landauer, L. M. Gomez, and S. T. Dumais, "Statistical Semantics: Analysis of the Potential Performance of Key-Word Information Systems," The Bell System Technical Journal, Vol. 62, No. 6, Part 3, July/August 1983, pp. 1753-1806.

Good, Michael D., John A. Whiteside, Dennis R. Wixon, and Sandra J. Jones, "Building a User-Derived Interface," Communications of the ACM, October 1984, Vol. 27, No. 10, pp. 1032-1043.

Hammer, Michael, "The Future of End-User Computing", Keynote Address at 1986 FOCUS Users Group, New Orleans, 1986.

Helander, G. S., "Improving System Usability for Business Professionals," IBM Systems Journal, Vol. 20, No. 3, 1981, pp. 294-305.

Intercomputer Communications Corp., Intercom 1500 PT 1500 Terminal Emulation for B2x Micros, Intercomputer Communications Corp., Cincinnati, Ohio, 1986.

James, E. B., "The User Interface," The Computer Journal, Vol. 23, No. 1, February 1980, pp. 25-28.

Karhan, C. J., C. A. Riley, M. S. Schoeffler, "Designing and Evaluating Standard Instructions for Public Telephones," The Bell System Technical Journal, Vol. 62, No. 6, Part 3, July/August 1983, pp. 1827-1848.

Kochan, Stephen G. and Patrick H. Wood, UNIX Shell Programming, Hayden Book Company, Hasbrouck Heights, New Jersey, 1985.

Krakowsky, P., "Searching for a UNIX User Interface," Attage, May/June 1985, pp. 36-37.

Landauer, T. K., S. T. Dumais, L. M. Gomez, and G. W. Furnas, "Human Factors in Data Access," The Bell System Technical Journal, November 1982, Vol. 61, No. 9, Part 2, pp. 2487-2510.

Martin, James, Design of Man-Computer Dialogues, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.

Meads, Jon A., "Friendly or Frivolous?", Datamation, April 1, 1985, pp. 96-100.

Peterson, James L., "A Note On Undetected Typing Errors," Communications of the ACM, July 1986, Vol. 29, No. 7, pp. 633-637.

Rushinek, Avi and Sara F. Rushinek, "What Makes Users Happy?", Communications of the ACM, July 1986, Vol. 29, No. 7, pp. 594-598.

Schwarz, Elmar, Ion P. Beldie, and Siegmund Pastoor, "A Comparison of Paging and Scrolling for Changing Screen Contents by Inexperienced Users," Human Factors, Vol. 25, No. 3, June 1983, pp. 265-282.

Shneiderman, Ben, Software Psychology: Human Factors in Computer Information Systems, Winthrop Publishers, Cambridge, Massachusetts, 1980.

Slator, Brian M., Matthew P. Anderson, and Walt Conley, "Pygmalion at the Interface," Communications of the ACM, July 1986, Vol. 29, No. 7, pp. 599-604.

Sobell, Mark G., "The Shell: Standard Input, Output, and Shell Scripts," Attage, August 1985, pp. 10-11.

Sobell, Mark G., "The Shell: Using Variables and Command Line Arguments," Attage, September 1985, pp. 12-14.

Sobell, Mark G., "The Shell: What Is It, How Does It Work?", Attage, July 1985, pp. 38-39.

Streeter, L. A., J. M. Ackroff, and G. A. Taylor, "On Abbreviating Command Names," The Bell System Technical Journal, Vol. 62, No. 6, Part 3, July/August 1983, pp. 1807-1826.

Thadhani, A. J., "Interactive User Productivity," IBM Systems Journal, Vol. 20, No. 4, 1981, pp. 407-423.

Tracz, William J., "Computer Programming and the Human Thought Process," Software - Practice and Experience, Vol. 9, 1979, pp. 127-137.

Voelcker, John, Paul Wallich, and Glenn Zorpette, "Personal Computers Part 2: Applications - Lessons Learned," IEEE Spectrum, May 1986, Vol. 23, No. 5, pp. 62-64.

Yavelberg, I. S., "Human Performance Engineering Considerations for Very Large Computer-Based Systems: The End User," The Bell System Technical Journal, May/June 1982, Vol. 61, No. 5, pp. 765-797.

APPENDICES

APPENDIX A

SignOn 5.0.4

Sat Oct 11, 1986 10 58 AM

BURROUGHS XE500 OPERATING SYSTEM BTOS B5 01

Selection | Enter an application name or leave this line blank
| | to display a Command form
|-----|

Password | Enter your assigned password (optional)
|-----|

Day/Date/Time | Enter the current day, date and time (if not already set)
|-----|

Then press the GO key.

User name (e.g., Allen)

Password

Date/Time (e.g., Fri Sep 9, 1983 8:00 am)

Executive 5.0.4 (OS t1C1strLfSp-5.0.4)

Path: [Sys](candidate)

User name: candidate
Sat Oct 11, 1986 11:00 AM

BURROUGHS XE500 OPERATING SYSTEM BTOS B5 01

!Selection ! Enter an application name or leave this line blank

! ! to display a Command form.

!Password ! Enter your assigned password (optional)

!Day/Date/Time ! Enter the current day, date and time (if not already set)

Then press the GO key.

User name (e.g., Allen)

candidate

Password

Date/Time (e.g., Fri Sep 9, 1983 8:00 am)

Command

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: candidate
Path: [Sys](candidate) Sat Oct 11, 1986 11:01 AM

 BURROUGHS XE500 OPERATING SYSTEM BTOS B5.01

! Selection ! Enter an application name or leave this line blank
! ! to display a Command form.

! Password ! Enter your assigned password (optional).

! Day/Date/Time ! Enter the current day, date and time (if not already set).

 Then press the GO key.

User name (e.g., Allen) candidate

Password

Date/Time (e.g., Fri Sep 9, 1983 8:00 am)

Command intercom 1500

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: candidate
Path: [Sys](candidate) Sat Oct 11, 1986 11:02 AM

Command intercom 1500
Intercom 1500
[[Interface (cluster, A, or B; default = cluster)]]
[Command file]

APPENDIX B

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: rsl
Path: [Sys] <RSL> Sat Oct 11, 1986 2:18 PM

Command

Commands are:

Append

Asynchronous Terminal Emulator

Backup Volume

Batch

Batch Status

Bootstrap

Change Page Format

Change Volume Name

Cluster Status

Console

Copy

Create Configuration File

Create Directory

Create File

Debug File

Delete

Dump

Edit

Enhanced Multiplan

Files

Floppy Copy

Format

Install Batch

Install Spooler

Intercom 1500

ISAM Configure

ISAM Copy

ISAM Create

ISAM Delete

ISAM Install

ISAM Rename

ISAM Reorganize

ISAM Set Protection

ISAM Status

IVArchive

IVolume

LCopy

Login

Logout

MAdmin Agent Status

Maintain File

Make Wheel Set

MBackup Volume

MBtos Config

Press NEXT PAGE or SCROLL UP to continue

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: rsl
 Path: [Sys](RSL) Sat Oct 11, 1986 2:19 PM

Format

- MCdtIO
- MChange Volume Name
- MCopy
- MCreate Configuration File
- MCreate Directory
- MCreate Partition
- MDelete
- MDisable Cluster
- MDisk Verify
- Merge
- MFiles
- MHistogram
- MInstall Server
- MIVolume
- MMaintain Files
- MMake Translation File
- Modem
- MPartition Status
- MPLog
- MRemove Directory
- MRemove Partition
- MRename

MBtos Config

- MRestore
- MResume Cluster
- MSelective Backup
- MSet Directory Protection
- MSet File Protection
- MSysload
- MTape Restore
- MVacate Partition
- MVersion
- MVolume Report
- MVolume Status
- New Command
- Path
- PLog
- Print
- R6k
- rbase
- Record
- Recover
- Remove Command
- Remove Directory
- Rename

Press NEXT PAGE or SCROLL UP to continue

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: rsl
 Path: [Sys](RSL) Sat Oct 11, 1986 2:19 PM

Merge	MVolume Report
MFiles	MVolume Status
MHistogram	New Command
Minstall Server	Path
MIVolume	PLog
MMaintain Files	Print
MMake Translation File	R6k
Modem	rbase
MPartition Status	Record
MPLog	Recover
MRemove Directory	Remove Command
MRemove Partition	Remove Directory
MRename	Rename
Replay	Set Time
Restore	Software Installation
Run	Sort
Run File	Spooler Status
Screen Dump	Stop Record
Screen Setup	Submit
Selective Backup	SWP
Set Directory Protection	Type
Set File Prefix	Unix
Set Protection	Volume Status

Command

APPENDIX C

Executive 5.0.4 (OS t1ClstrLfSp-5.0.4) User name: rsl
Path: [Sys]<RSL> Sat Oct 11, 1986 2:10 PM

Command

Append

Append each of a list of files to a (possibly existing) file.

Asynchronous Terminal Emulator

Allow a workstation to emulate a TTY.

Backup Volume

Archive all of a volume's files and verify the integrity of the volume.

Batch

Queue the specified JCL file in a batch queue.

Batch Status

Display the progress of batch jobs and queues.

Bootstrap

Bootstrap a specified diagnostic or operating system.

Change Page Format

Change the default page format of the word processor.

Change Volume Name

Change the name and password of a disk volume.
Press NEXT PAGE or SCROLL UP to continue

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: rsl
Path: [Sys](RSL) Sat Oct 11, 1986 2:11 PM

Change Volume Name
Change the name and password of a disk volume.

Cluster Status
Report status information about the activity on a communications line.

Console
pt-1500 login by console channel a

Copy
Copy a file to another file.

Create Configuration File
Create a printer or communications configuration file.

Create Directory
Create a new directory on a disk volume.

Create File
Creates a file (contents are undefined)

Debug File
Examine and modify the data in files and devices.

Delete
Press NEXT PAGE or SCROLL UP to continue

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: rsl
Path: [Sys]<RSL> Sat Oct 11, 1986 2:12 PM

Delete

Delete each of a list of files.

Dump

Display contents in hexadecimal and ASCII or compare two files.

Edit

Invoke the Editor.

Enhanced Multiplan

Business Planning Application (Enhanced)

Files

Display information about each of a list of files.

Floppy Copy

Duplicate floppy diskettes.

Format

Format each of a list of files.

Install Batch

Create a secondary partition and install a Batch Manager in it.

Press NEXT PAGE or SCROLL UP to continue

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: rsl
Path: [Sys] (RSL) Sat Oct 11, 1986 2:12 PM

Create a secondary partition and install a Batch Manager in it.

Install Spooler
Install and invoke the printer spooler.

Intercom 1500
PT1500 Emulator

ISAM Configure
Create or change a configuration file to be used by ISAM Install.

ISAM Copy
Copy the files of an ISAM data set, producing a new ISAM data set.

ISAM Create
Create an empty ISAM data set with the specified record size and index fields.

ISAM Delete
Delete both files of an ISAM data set, destroying all data in the data set.

ISAM Install
Install the ISAM multi-user access package in memory.

ISAM Rename
Rename the files of an ISAM data set, producing an ISAM data set.
Press NEXT PAGE or SCROLL UP to continue

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: rsl
Path: [Sys](RSL) Sat Oct 11, 1986 2:13 PM

ISAM Rename
Rename the files of an ISAM data set, producing an ISAM data set.

ISAM Reorganize
Build an ISAM data set from a file.

ISAM Set Protection
Change the passwords used to gain access to an existing ISAM data set.

ISAM Status
Display information about an ISAM data set.

IVArchive
Initialize the floppy in drive F0 as an archive volume.

IVolume

LCopy
Copy a list of files.

Login
Set default volume name, directory name, file prefix, password, and node.

Logout
Press NEXT PAGE or SCROLL UP to continue

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: rsl
Path: [SysJ<RSL> Sat Oct 11, 1986 2:13 PM

Logout

Terminate the current user session.

MAdmin Agent Status

XE500 utility which displays the status of the Admin Agents

Maintain File

Verify and/or repair structures of RSAM and DAM files.

Make Wheel Set

Make a Print Wheel set for OFISwriter25.

MBackup Volume

XE500 utility which archives all files contained on a volume

MBtos Config

XE500 utility which configures the BTOS system environment

MCdtIO

XE500 utility which provides a remote CLI facility

MChange Volume Name

XE500 utility which changes the name and password of a volume

Press NEXT PAGE or SCROLL UP to continue

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: rsl
Path: [Sys]<RSL> Sat Oct 11, 1986 2:14 PM

XE500 utility which changes the name and password of a volume

MCopy

XE500 utility which copies a file to another file

MCreate Configuration File

XE500 utility which creates a configuration file

MCreate Directory

XE500 utility which creates a new directory on a volume

MCreate Partition

XE500 utility which creates a partition on an XE500 processor

MDelete

XE500 utility which deletes each of a list of files

MDisable Cluster

XE500 utility which disables cluster operations

MDisk Verify

XE500 utility which performs disk verification

Merge

Merge several preexisting files of records
Press NEXT PAGE or SCROLL UP to continue

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: rsl
Path. [Sys]<RSL> Sat Oct 11, 1986 2:14 PM

Merge
Merge several preexisting files of records

MFiles
XE500 utility which displays status information about each of a list of files

MHistogram
XE500 utility which statistically samples the program counter

MInstall Server
XE500 utility which loads a secondary partition

MIVolume
XE500 utility which initializes a disk volume

MMaintain Files
XE500 utility which modifies and reads data files

MMake Translation File
XE500 utility which generates a translation file

Modem
Invoke A.T.E. 1200 baud to modem on channel A

MPartition Status
Press NEXT PAGE or SCROLL UP to continue

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: rsl
Path: [SysJ(RSL) Sat Oct 11, 1986 2:15 PM

MPartition Status
XE500 utility which displays partition status information

MPLog
XE500 utility which displays the system log

MRemove Directory
XE500 utility which removes a directory

MRemove Partition
XE500 utility which removes an XE500 partition

MRename
XE500 utility which renames a file

MRestore
XE500 utility which restores previously archived files

MResume Cluster
XE500 utility which re-enables cluster operations

MSelective Backup
XE500 utility which copies selected files to an archive file

Press NEXT PAGE or SCROLL UP to continue

```

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4)
Path: [Sys]<RSL>
User name: rsl
Sat Oct 11, 1986 2:15 PM
-----
XE500 utility which copies selected files to an archive file

MSet Directory Protection
XE500 utility which changes a directory's password or default file protection level
e1

MSet File Protection
XE500 utility which assigns a new protection level (and password) to a file

MSysload
XE500 utility which interactively loads XE500 software

MTape Restore
XE500 utility which restores the files previously archived on a tape

MVacate Partition
XE500 utility which terminates all tasks within a partition

MVersion
XE500 utility which displays the version level of a file

MVolume Report
XE500 utility which displays volume bad spot information

MVolume Status
Press NEXT PAGE or SCROLL UP to continue

```

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: rsl
Path: [Sys]<RSL> Sat Oct 11, 1986 2:16 PM

MVolume Status
XE500 utility which displays the status of a volume

New Command
Add a new command to those recognized by the Executive.

Path
Set default volume name, directory name, file prefix, password, and node.

PLog
Print the system error log.

Print
Add the files to the printer spooler queue.

R6k
Invoke RBase 6000 Data Base

rbase

Record
Record keystrokes in a file. Replay file later with the Submit command.

Press NEXT PAGE or SCROLL UP to continue

Executive 5.0.4 (OS t1ClstrLfSp-5.0.4) User name: rsl
Path: [Sys]<RSL> Sat Oct 11, 1986 2:16 PM

Record keystrokes in a file. Replay file later with the Submit command.

Recover

Recover your last OFISwriter25 session.

Remove Command

Remove a command from those recognized by the Executive

Remove Directory

Remove a directory from a disk.

Rename

Give an existing file a new name.

Replay

Replay an editing session.

Restore

Restore previously backed up files.

Run

Run a run file with parameters

Run File

Invoke a user program by specifying its run file
Press NEXT PAGE or SCROLL UP to continue

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: rsl
Path: [Sys]<RSL> Sat Oct 11, 1986 2:17 PM

Run File

Invoke a user program by specifying its run file

Screen Dump

Dump the screen contents to the specified printer.

Screen Setup

Change one or more of the video display attributes.

Selective Backup

Copy selected files to archive volume.

Set Directory Protection

Change a directory's password or default file protection level.

Set File Prefix

Set the default file prefix when vol name and dir name are omitted.

Set Protection

Assign new protection level (and password) to each of a list of files.

Set Time

Set the system clock for the entire cluster.

Software Installation

Press NEXT PAGE or SCROLL UP to continue

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: rsl
Path: [Sys]<RSL> Sat Oct 11, 1986 2:17 PM

Software Installation
Install software from a Burroughs floppy disk package

Sort
Sort one or more files of data records.

Spooler Status
Report status information about the printer spooler.

Stop Record
Stop recording keystrokes in a command file.

Submit
Read keystrokes from a command file rather than the keyboard.

SWP
Invoke OFISwriter25 Secretarial Word Processor.

Type
Display one or more files on the video display.

Unix

Press NEXT PAGE or SCROLL UP to continue

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: rsl
Path: [Sys](RSL) Sat Oct 11, 1986 2:18 PM

Sort

Sort one or more files of data records.

Spooler Status

Report status information about the printer spooler.

Stop Record

Stop recording keystrokes in a command file.

Submit

Read keystrokes from a command file rather than the keyboard.

SWP

Invoke OFISwriter25 Secretarial Word Processor.

Type

Display one or more files on the video display.

Unix

Volume Status

Display the status of a volume.

Command

APPENDIX D

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4) User name: rsl
Path: [Sys]<candidate> Sat Nov 22, 1986 6:49 PM

Command create file
Create File
File name
[Volume or Directory password]
[File password]
[File protection level (default = 15)]
[Size in sectors (default = 0)]
[Overwrite ok?]

APPENDIX E

3:42 PM

.profile

Nov 22, 1986

```
PS1="COMMAND: "  
PATH=:/bin:/etc:/usr/bin:/usr/local/bin:/usr/include:/db/ingres/b  
in:/civ/rsl/bin:/usr/oa
```

APPENDIX F

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4)
Path: [Sys]<Sys>

Tue Jan 6, 1987 6:12 PM

Command append
Append
File list from
File to
[Confirm each?]

Command copy
Copy
File from
File to
[Overwrite ok?]
[Confirm each?]

Command create directory
Create Directory
New directory name
[Protection level (default = 15)]
[Maximum number of files (default = 45)]
[Password for new directory]
[Volume password]

Command

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4)
Path: [Sys]<Sys>

Tue Jan 6, 1987 6:13 PM

Command

Command create file
Create File
File name
[Volume or Directory password]
[File password]
[File protection level (default = 15)]
[Size in sectors (default = 0)]
[Overwrite ok?]

Command delete
Delete
File list
[Confirm each?]

Command edit
Edit
File
[Your name]

Command

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4)

Path: [Sys]<Sys>

Tue Jan 6, 1987 6:14 PM

Command files

Files

[File list]

[Details?]

[Print file]

Command print

Print

File list

[Queue name (default = SPL)]

[Number of copies]

[Delete after printing?]

[Special forms name]

[Print wheel name]

[Printing mode]

[Align form?]

[After date/time]

[Security mode?]

[Priority]

[Confirm each?]

Command

Command

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4)
Path: [Sys]<Sys>

Tue Jan 6, 1987 6:15 PM

Command remove directory
Remove Directory
Old directory name
[Volume or directory password]
[Delete all files in directory?]
[Confirm each while deleting?]

Command rename
Rename
Old file name
New file name
[Overwrite ok?]
[Confirm each?]

Command set directory protection
Set Directory Protection
Directory name (e.g., sys)
[Volume or directory password]
[New protection level (e.g., 15)]
[New password]

Command

Executive 5.0.4 (OS t1ClstrLfsSp-5.0.4)

Path: [Sys]<Sys>

Tue Jan 6, 1987 6:15 PM

Set Protection

File list

New protection level (e.g., 15)

[New password]

[Confirm each?]

Command sort

Sort

Input files

Output file

Keys

[Stable sort?]

[Work File 1]

[Work File 2]

[Log file]

[Suppress confirmation?]

Command type

Type

File list

[Confirm each?]

Command

APPENDIX G

Jan 5 06:44 1987 append Page 1

```
# *****
# shell script name : append
# November 8, 1986
# R. Loffman

# this shell uses "cat" to imitate BTOS' "append"

# this uses cursor control routines written in C -
#     inverse
#     mvdwn
#     mvup
#     normal
#     rectangle
#     thisline

# *****

# appprmt is like BTOS, i.e., it prompts the user for input

appprmt () {
# called by maindriver

# trap user hitting delete key - don't want to be
# left in inverse mode

trap "normal; mvdwn 4; exit" 2

mvdwn
echo Append

mvdwn 1
echo '  File list from      \c'
mvdwn 1
thisline 1
echo '  File to              \c'
mvdwn 1
thisline 1
echo '  [Confirm each?]      \c'

mvup 2
inverse
scrn
read files
resetscr

inverse
scrn
read tofile
resetscr

inverse
scrn
read confirm
resetscr

thisline 1
```

Jan 5 06:44 1987 append Page 2

```
# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0

# if confirm is null, user chose default
if [ -z "$confirm" ]
then
    valid=1
    confirm=0
fi

# does user want to confirm each ?
while [ "$valid" -eq 0 ]
do
    case $confirm in

# user wants to confirm each
    y | Y | yes | Yes ) valid=1
                        confirm=1 ;;

# user does not want to confirm each
    n | N | no | No ) valid=1
                     confirm=0 ;;

# invalid answer
    * ) echo $confirm 'is neither Yes nor No, confirm each? \c '
        mvup 1
        inverse
        scrn
        read confirm
        resetscr
        thisline 1

# user entered carriage return, default value
        if [ -z "$confirm" ]
        then
            valid=1
            confirm=0
        fi

# user entered something else, loop back up to case to check it

    esac
done

# somefiles - indicates if there are files to be appended
# 0 - no          1 - yes
somefiles=0

# flg - indicates presence of files to be appended
# 0 - no          1 - yes
flg=0

mvdwn 3
```

Jan 5 06:44 1987 append Page 3

```
# read files to be appended
for file in $files
do
# a directory? yes
if [ -d $file ]
then
echo $file is a directory - can not append.
echo

# exist? no
elif [ ! -s $file ]
then
echo $file does not exist - can not append.
echo

# exists
# and is first file ==> first file has to be cat'ed into
# temporary to create temporary file
elif [ "$flg" -eq 0 ]
then

# user indicated wanted to confirm each, one at a time
if [ "$confirm" -eq 1 ]
then
echo "Append $file (y or n) ? \c"
valid=0
while [ "$valid" -eq 0 ]
do
read check
echo
case $check in

# ok to append
y) cat $file > /tmp/appt$
flg=1
somefiles=1
echo Appending $file ... done.
echo
valid=1 ;;

# don't append
n) valid=1 ;;

# invalid response
*) echo $response 'is neither Yes nor No, reenter \c ' ;;
esac
done

# user does not want to confirm each, so append each in turn
else
cat $file > /tmp/appt$
flg=1
somefiles=1
echo Appending $file ... done.
echo
fi
```

Jan 5 06:44 1987 append Page 4

```
# exist
# not first file, so just attach to others already in
# temporary file
else

# user wants to confirm each
if [ "$confirm" -eq 1 ]
then
    echo "Append $file (y or n) ? \c"
    valid=0
    while [ "$valid" -eq 0 ]
    do
        read check
        echo
        case $check in

# ok to append
y ) cat /tmp/app$$ $file > /tmp/app$$save
    cp /tmp/app$$save /tmp/app$$
    flg=1
    somefiles=1
    echo Appending $file ... done
    echo
    valid=1 ;;

# don't append
n ) valid=1 ;;

# invalid response
* ) echo $response 'is neither Yes nor No, reenter \c ' ;;
    esac
done

else

# user does not want to confirm each, so append each in turn
cat /tmp/app$$ $file > /tmp/app$$save
cp /tmp/app$$save /tmp/app$$
echo Appending $file ... done
echo
fi
done

# were there files to be appended?
# no
if [ "$somefiles" -eq 0 ]
then echo All of the files were directories or did not exist.
    echo Append to $file did not occur.

# yes, but file to put them into is a directory
# will call function to create new file
elif [ -d $tofile ]
then
    echo $tofile is a directory - can not append into directory.
```

Jan 5 06:44 1987 append Page 5

```
newname

# yes and file to put them into already exists ==> overwrite it
elif [ -s $tofile ]
then
    cat $tofile /tmp/app$$ > /tmp/app$$save
    cp /tmp/app$$save $tofile
    echo Appended to $tofile.

# yes and file to put them does not exist, will create it
else
    cp /tmp/app$$ $tofile
    echo $tofile created.
fi

)
# end appprmt

scrn() {
# resets the rectangle for reading input
# called by appprmt and resetscr

thisline 21
rectangle 1 50
}
# end scrn

resetscr () {
#called by appprmt

normal
mvup
scrn
mvdwn 1
}
# end resetscr

newname () {
# called by appprmt
# creates new file when directory is the file where appended
# files are to be placed

echo 'Enter new file to be created: \c'
read name
echo

# valid - indicates if is directory name
# 0 - yes          1 - no
valid=0
while [ "$valid" -eq 0 ]
do

# input name is directory, reject it
if [ -d "$name" ]
then
```

Jan 5 06:44 1987 append Page 6

```
    echo $name 'is directory, re enter file name: \c'
    read name
    echo

# input name is existing file, overwrite it
elif [ -s "$name" ]
then
    cat $name /tmp/app$$ > /tmp/app$$ save
    cp /tmp/app$$ save $name
    valid=1
    echo $name created

# input name is new file, create it
else
    cp /tmp/app$$ $name
    valid=1
    echo $name created.
fi
done
)

# append - MAIN DRIVER

# test for missing argument
# if argument missing, assume user wants to be prompted for input
if [ $# -eq 0 ]
then
    appprmt
    exit
fi

# if user supplies argument(s), assume user does not want to be
# prompted for input.
# validate user supplied input

# one argument - not enough
if [ $# -eq 1 ]
then
    echo Not enough files listed - reenter.
    exit
fi

# no files to start with
somefiles=0

# no files to put in yet
flg=0
while [ "$#" -ne 1 ]
do

# file a directory?
# yes - don't append it
if [ -d $1 ]
then
    echo $1 is directory, can not append
```

Jan 5 06:44 1987 append Page 7

```
# file exist?
# no - append doesn't make sense
  elif [ ! -s $1 ]
  then
    echo $1 does not exist, can not append.

# ok to append
# first file ==> create temporary file
  elif [ "$flg" -eq 0 ]
  then
    cat $1 > /tmp/app$$
    flg=1
    somefiles=1
    echo Appending $1 ... done.

# ok to append
# not first file ==> add to ones already there
  else
    cat /tmp/app$$ $1 > /tmp/app$$save
    cp /tmp/app$$save /tmp/app$$
    echo Appending $1 ... done.
  fi

# get next file in list
  shift
done

# leave last file as file to append everything to

# all of the files were nonexistent or directories
# nothing to append
  if [ "$somefiles" -eq 0 ]
  then echo All of the files were directories or did not exist.
    echo Append did not occur.

# file to contain appended files is a directory
# create new file to put them into
  elif [ -d $1 ]
  then
    echo $1 is a directory, can not append into directory.
    newname

# file exists ==> overwrite it
  elif [ -s $1 ]
  then
    cat $1 /tmp/app$$ > /tmp/app$$save
    cp /tmp/app$$save $1
    echo Appended to $1.

# file doesn't exist ==> create it
  else
    cp /tmp/app$$ $1
    echo $1 created.
  fi
fi
```

Jan 5 06:54 1987 copy Page 1

```
# *****
# shell script name : copy
# November 13, 1986
# R. Loffman

# this shell uses "cp" to imitate BTOS' "copy"

# this shell uses cursor control routines written in C -
#      inverse
#      mvdwn
#      mvup
#      normal
#      rectangle
#      thisline

# *****
copyprmt () {
# copyprmt is like BTOS, i.e., it prompts the user for input

# trap the user hitting delete key - don't want to be left in inverse

trap "normal; mvdwn 4; exit" 2

mvdwn 1
echo Copy

mvdwn 1
echo ' File from          \c'
mvdwn 1
thisline 1
echo ' File to           \c'
mvdwn 1
thisline 1
echo ' [Overwrite ok?]    \c'
mvdwn 1
thisline 1
echo ' [Confirm each?]    \c'

mvup 3
inverse
scrn
read fromfile
resetscr

inverse
scrn
read tofile
resetscr

inverse
scrn
read over
resetscr

inverse
scrn
```

Jan 5 06:54 1987 copy Page 2

```
read confirm
resetscr
```

```
thisline 1
```

```
# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0
```

```
# if over is null, user chose default
if [ -z "$over" ]
then
    valid=1
    over=0
fi
```

```
# does user want to overwrite?
while [ "$valid" -eq 0 ]
do
    case $over in
```

```
# user wants to overwrite
y | Y | yes | Yes ) valid=1
                    over=1 ;;
```

```
# user does not want to overwrite
n | N | no | No ) valid=1
                  over=0 ;;
```

```
# invalid answer
* ) echo $over 'is neither Yes nor No, overwrite ok? \c '
    read over
```

```
# user entered carriage return, default value
if [ -z "$over" ]
then
    valid=1
    over=0
fi
```

```
# user entered something else, loop back up to case to check it
```

```
esac
done
```

```
# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0
```

```
# if confirm is null, user chose default
if [ -z "$confirm" ]
then
    valid=1
    confirm=0
fi
```

Jan 5 06:54 1987 copy Page 3

```
# does user want to confirm each ?
while [ "$valid" -eq 0 ]
do
    case $confirm in

# user wants to confirm each
    y | Y | yes | Yes ) valid=1
                        confirm=1 ;;

# user does not want to confirm each
    n | N | no | No ) valid=1
                     confirm=0 ;;

# invalid answer
    * ) echo $confirm 'is neither Yes nor No, confirm each? \c'
        read confirm

# user entered carriage return, default value
    if [ -z "$confirm" ]
    then
        valid=1
        confirm=0
    fi

    esac
done

# is the file to be copied a directory? yes
if [ -d $fromfile ]
then
    echo $fromfile is a directory ... can not copy.
    exit
fi

# exist? no
if [ ! -s $fromfile ]
then
    echo $fromfile: no such file ... can not copy.
    exit
fi

# does tofile exist? no
if [ ! -s $tofile ]
then

# does user want to confirm? yes
    if [ "$confirm" -eq 1 ]
    then
        echo "Copy $fromfile to $tofile? \c"
        valid=0
        while [ "$valid" -eq 0 ]
        do
            read check
            echo
            case $check in
```

Jan 5 06:54 1987 copy Page 4

```
#      ok to copy
      y) prnover
      valid=1 ;;

#      don't copy
      n ) echo Did not copy $fromfile to $tofile.
      valid=1 ;;

#      invalid response
      * ) echo $response 'is neither Yes nor No, reenter \c ' ;;
      esac
done

# user doesn't want to confirm
else
  prnover
fi

# tofile exists
else

# does user want to confirm each? yes
if [ "$confirm" -eq 1 ]
then

  echo "Copy $fromfile to $tofile? \c"
  valid=0
  while [ "$valid" -eq 0 ]
  do
    read check
    echo
    case $check in

#      ok to copy
      y) if [ "$over" -eq 1 ]
          then
            prover
            valid=1
          else
            echo Did not copy $fromfile, $tofile already exists.
            valid=1
          fi ;;

#      don't copy
      n ) echo Did not copy $fromfile to $tofile.
      valid=1 ;;

#      invalid response
      * ) echo $response 'is neither Yes nor No, reenter \c ' ;;
      esac
done

# user doesn't want to confirm
else

# does user want to overwrite? yes
```

Jan 5 06:54 1987 copy Page 5

```
    if [ "$over" -eq 1 ]
    then
        prover

# user doesn't want to overwrite
    else
        echo Did not copy $fromfile, $tofile already exists.
    fi
fi

# end big if
fi
}
# end copyprmt

scrn () {
# resets the rectangle for reading input
# called from copyprmt and resetscr

thisline 21
rectangle 1 50
}
# end scrn

resetscr () {
# called by copyprmt

normal
mvup
scrn
mvdwn 1
}
# end resetscr

prnover () {
# called from copyprmt

cp $fromfile $tofile
echo Copying $fromfile to $tofile ... done.
}

prover () {
# called from copyprmt

cp $fromfile $tofile
echo Copying $fromfile to $tofile \((overwriting\) ... done.
}

# copy - MAIN DRIVER

# test for missing argument
# if argument missing, assume user wants to be prompted for input
if [ $# -eq 0 ]
then
    copyprmt
```

```
    exit
fi

# if user supplies argument(s), assume user does not want to be
# prompted for input.
# validate user supplied input

# one argument - not enough
if [ $# -eq 1 ]
then
    echo Not enough files listed - reenter.
    exit
fi

# more than two arguments - too many
if [ $# -gt 2 ]
then
    echo More than two files listed - reenter.
    exit
fi

# first file a directory?

# yes - can not do copy
if [ -d $1 ]
then
    echo $1 is directory - can not copy.
    exit
fi

# does first file exist?

# no - copy with it does not make sense
if [ ! -s $1 ]
then
    echo $1: no such file - can not copy.
    exit
fi

# first file exists

# is second file a directory?
# yes
if [ -d $2 ]
then
    cp $1 $2
    echo Copying $1 to $2 ... done.

# does the second file exist?
# no - create it
elif [ ! -s $2 ]
then
    cp $1 $2
    echo Copying $1 to $2 ... done.
    exit
```

Jan 5 06:54 1987 copy Page 7

```
# second file exists, do not overwrite
else
    echo Did not copy $1, $2 already exists.
fi
```

Jan 5 16:58 1987 create Page 1

```
# *****
# shell script name : create
# November 14, 1986
# R. Loffman
```

```
# this shell uses "echo" to imitate BTOS' "create file"
#               "mkdir" to imitate BTOS' "create directory"
```

```
# this shell uses cursor control routines written in C -
#       inverse
#       mvdwn
#       mvup
#       normal
#       rectangle
#       thisline
```

```
# *****
```

```
crfile () {
# called by main driver
```

```
# is file to be created a directory?
# yes - then can not create as a file
if [ -d $name ]
then
    echo $name is a directory name - can not be used as a file name
    exit
fi
```

```
# does the file exist?
# no - create it
if [ ! -s $name ]
then
    echo "" > $name
    echo Creating file $name ... done
    exit
fi
```

```
# file exists - overwrite?
```

```
# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0
```

```
echo File $name 'already exists, overwrite? \c'
read over
```

```
# does user want to overwrite?
while [ "$valid" -eq 0 ]
do
    case $over in
```

```
# user wants to overwrite
y : Y : yes : Yes ) valid=1
    echo Creating file $name \((overwriting\) ... done. ;;
```

Jan 5 16:58 1987 create Page 2

```
# user does not want to overwrite
n : N : no : No ) valid=1
    echo Did not create file $name, already exists. ;;

# invalid answer
* ) echo 'Overwrite response invalid, overwrite ok? \c '
    read over

    esac
done

}
# end crfile

crfpmt () {
# called by main driver

#trap user hitting delete key - don't want to be left in inverse mode

trap "normal; mvdown 4; exit " 2

mvdown
echo Create file

mvdown 1
echo ' File name          \c'
mvdown 1
thisline 1
echo ' [Overwrite ok?]    \c'

mvup 1
inverse
scrn
read name
resetscr

inverse
scrn
read over
resetscr

thisline 1

# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0

# if over is null, user chose default
if [ -z "$over" ]
then
    valid=1
    over=0
fi

# does user want to overwrite?
```

Jan 5 16:58 1987 create Page 3

```
while [ "$valid" -eq 0 ]
do
    case $over in

# user wants to overwrite
    y | Y | yes | Yes ) valid=1
                        over=1 ;;

# user does not want to overwrite
    n | N | no | No ) valid=1
                    over=0 ;;

# invalid answer
    * ) echo 'Overwrite response invalid, overwrite ok? \c ' a
        mvup 1
        inverse
        scrn
        read over
        resetscr
        thisline 1

# user entered carriage return, default value
        if [ -z "$over" ]
        then
            valid=1
            over=0
        fi

# user entered something else, loop back up to check it
    esac
done

mvdown 3

# is file to be created a directory?
# yes - then can not create as a file
if [ -d $name ]
then
    echo $name is a directory name - can not be used as a file name
    exit
fi

# does the file exist?
# no - create it
if [ ! -s $name ]
then
    `echo "" > $name`
    echo Creating file $name ... done.
    exit
fi

# file exists

# did user select overwrite?
# user said overwrite
if [ "$over" -eq 1 ]
```

Jan 5 16:58 1987 create Page 4

```
then
    `echo "" > $name`
    echo Creating file $name \((overwriting\) ... done.
    exit
fi

# user doesn't want to overwrite
echo Did not create file $name, already exists.

)

# end crfpmt

crdir () {
    # called by main driver and crdpmt

    # check that directory does not already exist
    # exists as a directory
    if [ -d $name ]
    then
        echo $name directory already exists.
        exit
    fi

    # does not exist - create it
    mkdir $name
    echo Creating directory $name ... done.

}

# end crdir

crdpmt () {
    # called by main driver

    #trap user hitting delete key - don't want to be left in inverse mode

    trap "normal; mvdown 4; exit " 2

    mvdown
    echo Create directory

    mvdown 1
    # prompt for directory name
    echo ' New directory name      \c'

    inverse
    scrn
    read name
    resetscr

    thisline 1
    # call function to create directory
    crdir
}

# end crdpmt
```

Jan 5 16:58 1987 create Page 5

```
scrn () {
# resets the rectangle for reading input

thisline 22
rectangle 1 50
}
# end scrn

resetscr () {

normal
mvup
scrn
mvdwn 1
}
# end resetscr

# create - MAIN DRIVER

# if argument missing
if [ $# -eq 0 ]
then
    echo Did not specify file or directory
    exit
fi

# if number of arguments is more than 2, invalid use of command
if [ $# -gt 2 ]
then
    echo 'Too many arguments - reenter command.'
    exit
fi

# if first argument is "file" - user wants to create file
if [ "$1" = file ]
then
    # if only 2 arguments - file and a name - user does not want prompting
    if [ $# -eq 2 ]
    then
        name=$2
        crfile
        exit
    # user wants prompting
    else
        crfpmt
        exit
    fi
fi

# if first argument is "directory" - user wants to create directory
if [ "$1" = directory ]
then
    # if only 2 arguments - directory and a name - user does not want prompt
```

Jan 5 16:58 1987 create Page 6

```
if [ $# -eq 2 ]
then
    name=$2
    crdir
    exit

# user wants prompting
else
    crdpmt
    exit
fi

# invalid use of create command
echo No such command - create $1 - reenter command
```

Jan 5 16:59 1987 delete Page 1

```
# *****
# shell script name : delete
# November 17, 1986
# R. Loffman

# this shell uses "rm" to imitate BTOS' "delete"

# this shell uses cursor control routines written in C -
#     inverse
#     mvdown
#     mvup
#     normal
#     rectangle
#     thisline

# *****

# delprmt is like BTOS, i.e., it prompts the user for input

delprmt () {

#trap user hitting delete key - don't want to be left in inverse mode

trap "normal; mvdown 4; exit" 2

mvdown
echo Delete

mvdown 1
thisline 1
echo '  File list          \c'
mvdown 1
thisline 1
echo '  [Confirm each?]    \c'

mvup 1
inverse
scrn
read files
resetscr

inverse
scrn
read confirm
resetscr

thisline 1

# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0

# if confirm is null, user entered default
if [ -z "$confirm" ]
then
    valid=1
```

Jan 5 16:59 1987 delete Page 2

```
confirm=0
fi

# does user want to confirm each?
while [ "$valid" -eq 0 ]
do
    case $confirm in

# user wants to confirm each
    y | Y | yes | Yes ) valid=1
                        confirm=1 ;;

# user does not want to confirm each
    n | N | no | No ) valid=1
                    confirm=0 ;;

# invalid answer
    * ) echo $confirm 'is neither Yes nor No, confirm each? \c '
        mvup 1
        inverse
        scrn
        read confirm
        resetscr
        thisline 1

# user entered carriage return, default value
        if [ -z "$confirm" ]
        then
            valid=1
            confirm=0
        fi

# user entered something else, loop back up to case to check it

    esac
done

mvdown 3

# read files to be deleted
for file in $files
do
# a directory? yes
    if [ -d $file ]
    then
        echo $file is a directory - can not delete.
        echo

# exist? no
    elif [ ! -s $file ]
    then
        echo $file does not exist - can not delete.
        echo

# exists
```

Jan 5 16:59 1987 delete Page 3

```
# user indicated wanted to confirm each, one at a time
elif [ "$confirm" -eq 1 ]
then
    echo "Delete $file (y or n) ? \c"
    valid=0
    while [ "$valid" -eq 0 ]
    do
        read check
        echo
        case $check in
            #      ok to delete
            y) rm $file
                echo Deleting $file ... done.
                echo
                valid=1 ;;
            #      don't delete
            n ) valid=1 ;;
            #      invalid response
            * ) echo $response 'is neither Yes nor No, reenter \c ' ;;
        esac
    done

# user does not want to confirm each, so delete each in turn
else
    rm $file
    echo Deleting $file ... done.
    echo
fi
done
}
# end delprmt

scrn () {
# resets the rectangle for reading input

thisline 21
rectangle 1 50

}
# end scrn

resetscr () {
normal
mvup
scrn
mvdwn 1
}
# end resetscr

# delete - MAIN DRIVER

# test for missing argument
```

Jan 5 16:59 1987 delete Page 4

```
# if argument missing, assume user wants to be prompted for input
if [ $# -eq 0 ]
then
    delprmt
    exit
fi

# if user supplies argument(s), assume user does not want to be
# prompted for input.
# validate user supplied input

# need to shift thru all of them

while [ "$#" -ne 0 ]
do

#   file a directory?
#   yes - don't delete it
    if [ -d $1 ]
    then
        echo $1 is directory, can not delete.

#   file exist?
#   no - delete doesn't make sense
    elif [ ! -s $1 ]
    then
        echo $1 does not exist, can not delete.

#   ok to delete
    else
        rm $1
        echo Deleting $1 ... done.
    fi

#   get next file in list
    shift
done
```

Jan 5 17:00 1987 edit Page 1

```
# *****
# shell script name : edit
# November 17, 1986
# R. Loffman

# this shell uses "vi" to imitate BTOS' "edit"
# expects the user to know vi

# because there is already a system command edit, the system command
# will be the one that is executed instead of this version of edit.

# to use this version of edit, either
# 1. specify full path name to this edit or
# 2. make the path of the user include the path to the
#    directory where this is located and place this path
#    before the path to the system's edit. THIS IS NOT ADVISED

# this uses cursor control routines written in C -
#     inverse
#     mvdwn
#     mvup
#     normal
#     rectangle
#     thisline

# *****
editprmt () {
# editprmt is like BTOS, i.e., it prompts the user for input

#trap the user hitting the delete key - don't want to be left in inverse

trap "normal; mvdwn 4; exit;" 2

mvdwn
echo Edit

mvdwn 1
echo '  File          \c'

inverse
scrn
read file
resetscr

thisline 1

# is the file to be edited a directory? yes
if [ -d $file ]
then
    echo $file is a directory ... can not edit.
    exit
fi

# note: if file doesn't exist, vi will create it
vi $file
}
```

Jan 5 17:00 1987 edit Page 2

```
# end editprmt

scrn () {

thisline 21
rectangle 1 30

}
# end scrn

resetscr () {

normal
mvup
scrn
mvdwn 1
}
# end resetscr

# edit - MAIN DRIVER

# test for missing argument
# if argument missing, assume user wants to be prompted for input
if [ $# -eq 0 ]
then
    editprmt
    exit
fi

# if user supplies argument(), assume user does not want to be
# prompted for input.
# validate user supplied input

# more than one argument - to many
if [ $# -ne 1 ]
then
    echo Can only edit one file at a time - reenter.
    exit
fi

# file a directory?

# yes - can not do edit
if [ -d $1 ]
then
    echo $1 is directory - can not edit.
    exit
fi

vi $1
```

```
# *****
# shell script name : files
# November 17, 1986
# R. Loffman

# this shell uses "ls" to imitate BTOS' "files"

# this shell uses cursor control routines written in C -
#     inverse
#     mvdown
#     mvup
#     normal
#     rectangle
#     thisline

# *****

# filesprmt is like BTOS, i.e., it prompts the user for input

filesprmt () {
# called by main criver

# trap the user hitting the delete key - don't want to be left in inverse
# mode

trap "normal; mvdown 4; exit" 2

mvdown
echo Files

mvdown 1
echo '  File list           \c'
mvdown 1
thisline 1
echo '  [Details each?]     \c'
mvdown 1
thisline 1
echo '  [Print file]         \c'

mvup 2
inverse
scrn
read files
resetscr

inverse
scrn
read detail
resetscr

inverse
scrn
read queue
resetscr

thisline 1
```

```
# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0
# if detail is null, user chose default
if [ -z "$detail" ]
then
    valid=1
    detail=0
fi

# does user want details of each?
while [ "$valid" -eq 0 ]
do
    case $detail in

# user wants details of each
    y | Y | yes | Yes ) valid=1
                        detail=1 ;;

# user does not want details of each
    n | N | no | No ) valid=1
                     detail=0 ;;

# invalid answer
    * ) echo $detail 'is neither Yes nor No, details each? \c '
        read detail

# user entered carriage return, default value
    if [ -z "$detail" ]
    then
        valid=1
        detail=0
    fi

# user entered something else, loop back up to case to check it
    esac
done

# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0
# if queue is null, user chose default
if [ -z "$queue" ]
then
    valid=1
    queue=0
fi

# does user want to print each ?
while [ "$valid" -eq 0 ]
do
    case $queue in

# user wants to print each
```

```
spl : SPLB ) valid=1
      queue=SPL ;;

# user does not want to print each
splb : SPLB ) valid=1
      queue=SPLB ;;

# invalid answer
* ) echo $queue 'is neither SPL nor SPLB nor blank, reenter \c '
  read queue

# user entered carriage return, default value
if [ -z "$queue" ]
then
  valid=1
  queue=0
fi

# user entered something else, loop back up to case to check it
esac
done

# flg - indicates presence of files to be listed
# 0 - no          1 - yes
flg=0

# read files to be listed
for file in $files
do

# user wants details about files
if [ "$detail" -eq 1 ]
then
# does file exist? no
if [ ! -s $file ]
then
  echo $file does not exist - can not list it.

# file exists - is it a directory?
# yes
elif [ -d $file ]
then
  echo $file is a directory:
  ls -al $file
  echo End of $file files.
  flg=1

# file exists and is not a directory
else
  ls -al $file
  prn
  flg=1
fi

# user doesn't want details
else
```

```
# does file exist? no
if [ ! -s $file ]
then
    echo $file does not exist - can not list it.

# file exists - is it a directory?
# yes
elif [ -d $file ]
then
    echo $file is a directory
    ls $file
    echo End of $file files
    flg=1

# file exists and is not a directory
else
    ls $file
    prn
    flg=1
fi
done
nofile

)
# end filesprmt

prn () {
# called by filesprmt
# does printing to the specified queue

if [ "$queue" = SPL ]
then
    lpr -q $queue $file
    fi

if [ "$queue" = SPLB ]
then
    lpr -q SPLB $file
    fi

}
# end prn

nofile (){
# called by main driver and filesprmt

# were there files to be listed?
# no
if [ "$flg" -eq 0 ]
then echo All of the files did not exist
    echo Files did not occur.
fi
}
# end nofile
```

```
scrn () {
# resets the rectangle for reading input

thisline 21
rectangle 1 50

}
# end scrn

resetscr () {

normal
mvup
scrn
mvdwn 1
}
# end resetscr

# files - MAIN DRIVER

# test for missing argument
# if argument missing, assume user wants to be prompted for input
if [ $# -eq 0 ]
then
    filesprmt
    exit
fi

# if user supplies argument(s), assume user does not want to be
# prompted for input.
# validate user supplied input

# no files yet
flg=0
while [ "$#" -ne 0 ]
do

#   file exist?
#   no - files doesn't make sense
    if [ ! -s $1 ]
    then
        echo $1 does not exist, can not list it.

#   ok to list it and is first file ==> set flag to show presence of file
    elif [ "$flg" -eq 0 ]
    then

#       is the file a directory?
#       yes
        if [ -d $1 ]
        then
            echo $1 a directory:
            ls $1
            echo End of $1 files.
```

```
        flg=1

#       file not a directory
        else
            ls $1
            flg=1
        fi

#       list file - not first file
        else

#           is the file a directory?
#           yes
            if [ -d $1 ]
            then
                echo $1 a directory:
                ls $1
                echo End of $1 files.

#           no
            else
                ls $1
            fi
        fi
    shift
done

nofile
```

Jan 5 17:03 1987 print Page 1

```
# *****
# shell script name : print
# November 20, 1986
# R. Loffman

# this shell uses "print" to imitate BTOS' "print"

# this shell uses cursor control routines written in C -
#     inverse
#     mvdwn
#     mvup
#     normal
#     rectangle
#     thisline

# because there is already a system command print, the system command
# will be the one that is executed instead of this version of print.

# to use this version of print, either
#     1. specify full path name to this print or
#     2. make the path of the user include the path to the
#         directory where this is located and place this path
#         before the path to the system's print.  THIS IS NOT ADVISED
# *****
prprmt () {
# prprmt is like BTOS, i.e., it prompts the user for input

# trap the user hitting the delete key - don't want to be left in inverse
trap "normal, mvdwn 4; exit" 2

mvdwn
echo Print

mvdwn
echo '  File list'                \c'

mvdwn 1
thisline 1
echo '  [Queue name (default = SPL)]'  \c'

mvdwn 1
thisline 1
echo '  [Number of copies]'          \c'

mvdwn 1
thisline 1
echo '  [Delete after printing?]'    \c'

mvdwn 1
thisline 1
echo '  [Confirm each?]'             \c'

mvup 4
inverse
scrn
```

Jan 5 17:03 1987 print Page 2

```
read files
resetscr
```

```
inverse
scrn
read queue
resetscr
```

```
inverse
scrn
read copies
resetscr
```

```
inverse
scrn
read delete
resetscr
```

```
inverse
scrn
read confirm
resetscr
```

```
thisline 1
```

```
# determine queue
```

```
# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0
```

```
# if queue is null, user chose default
if [ -z "$queue" ]
then
    valid=1
    queue=SPL
fi
```

```
while [ "$valid" -eq 0 ]
do
    case $queue in
```

```
# SPL
spl : SPL ) valid=1 ;;
```

```
# SPLB
splb : SPLB ) valid=1 ;;
```

```
# invalid answer
# ) echo 'Queue name invalid, reenter: \c '
    read queue
```

```
# user entered carriage return, default
if [ -z "$queue" ]
then
    valid=1
```

Jan 5 17:03 1987 print Page 3

```
        queue=SPL
    fi
# user entered something - loop back to case to check it
esac
done

# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0

# check if copies is null => user wants default
if [ -z "$copies" ]
then
    copies=1
    valid=1
fi

# check if user entered valid number
# done by checking if 1 > input ie., input is not a number
if [ 1 -gt "$copies" ]
then
    while [ "$valid" -eq 0 ]
    do

# invalid answer
echo 'Copies not numeric - reenter \'c\''

read copies

# user entered carriage return => default
if [ -z "$copies" ]
then
    valid=1
    copies=1

# user entered non numeric
elif [ 1 -gt "$copies" ]
then
    valid=0

# user entered valid number
else
    valid=1
fi
done
fi

# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0

# if delete is null, user chose default
if [ -z "$delete" ]
then
    valid=1
    delete=0
```

Jan 5 17:03 1987 print Page 4

```
fi

# does user want to delete?
while [ "$valid" -eq 0 ]
do
    case $delete in

# user wants to delete
        y | Y | yes | Yes ) valid=1
                                delete=1 ;;

# user does not want to delete
        n | N | no | No ) valid=1
                            delete=0 ;;

# invalid answer
        * ) echo 'Delete response invalid, delete each after printing? \c '
            read delete

# user entered carriage return, default
        if [ -z "$delete" ]
        then
            valid=1
            delete=0
        fi

# user entered something else, loop back up to case to check it
    esac
done

# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0

# if confirm is null, user chose default
if [ -z "$confirm" ]
then
    valid=1
    confirm=0
fi

# does user want to confirm each ?
while [ "$valid" -eq 0 ]
do
    case $confirm in

# user wants to confirm each
        y | Y | yes | Yes ) valid=1
                            confirm=1 ;;

# user does not want to confirm each
        n | N | no | No ) valid=1
                            confirm=0 ;;

# invalid answer
```

Jan 5 17:03 1987 print Page 5

```
* ) echo 'Confirm response invalid, confirm each? \c '
  read confirm

#      user entered carriage return, default value
  if [ -z "$confirm" ]
  then
    valid=1
    confirm=0
  fi

#      user entered something else, loop back up to case to check it
  esac
done

# read files to be printed
for file in $files
do
# a directory? yes
  if [ -d $file ]
  then
    echo $file is a directory - can not print it.

# exist? no
  elif [ ! -s $file ]
  then
    echo $file does not exist - can not print it.

# exists - yess
  else

#      user wants to confirm each
    if [ "$confirm" -eq 1 ]
    then
      echo "Print $file (y or n) ? \c"
      valid=0
      while [ "$valid" -eq 0 ]
      do
        read check
        echo
        case $check in

#          ok to print
          y ) prnt
              valid=1 ;;

#          don't print
          n ) valid=1 ;;

#          invalid response
          * ) echo $response 'is neither Yes nor No, reenter \c ' ;;
        esac
      done

    else
```

Jan 5 17:03 1987 print Page 6

```
#      user does not want to confirm each, so print each in turn
prnt
fi
fi
done
)
# end prprmt

prnt () {
# called by prprmt

if [ "$delete" -eq 1 ]
then
    prloop
    rm $file
    echo $file deleted.
else
    prloop
fi

)
# end prnt

prloop () {
# called by prnt

i=1
while [ "$i" -le "$copies" ]
do
    lpr -s -q $queue $file
    echo Printing $file ... done.
    i=`expr $i + 1 `
done
)
# end prloop

scrn () {
# resets the rectangle for reading input

thisline 32
rectangle 1 40

)
# end scrn

resetscr () {

normal
mvup
scrn
mvdwn 1
)
# end resetscr
```

Jan 5 17:03 1987 print Page 7

print - MAIN DRIVER

test for missing argument

if argument missing, assume user wants to be prompted for input

if [\$# -eq 0]

then

prprompt

exit

fi

if user supplies argument(s), assume user does not want to be

prompted for input.

validate user supplied input

no files to start with
somefiles=0

no files to put in yet
flg=0

while ["\$#" -ne 0]
do

file a directory?

yes - don't print it

if [-d \$1]

then

echo \$1 is directory, can not print it.

file exist?

no - print doesn't make sense

elif [! -s \$1]

then

echo \$1 does not exist, can not print it.

ok to print

elif ["\$flg" -eq 0]

then

lpr -s -q SPL \$1

flg=1

somefiles=1

echo Printing \$1 ... done.

ok to print

else

lpr -s -q SPL \$1

echo Printing \$1 ... done.

fi

get next file in list

shift

done

nothing to Print

if ["\$somefiles" -eq 0]

then echo All of the files were directories or did not exist

echo Print did not occur.

Jan 5 17:03 1987 print Page 8

fi

Feb 10 03:33 1987 remove Page 1

```
# *****
# shell script name : remove
# November 28, 1986
# R. Loffman

# this shell uses "rmdir" to imitate BTOS' "remove directory"

# this shell uses cursor control routines written in C -
#     inverse
#     mvdwn
#     mvup
#     normal
#     rectangle
#     thisline

# *****

# rmdprmt is like BTOS, i.e. it prompts the user for input

rmdprmt () {

# trap the user hitting delete key - don't want to be left in inverse mode

trap "normal, mvdwn 4, exit" 2

mvdwn 1
echo Remove directory

mvdwn 1
echo ' Old directory name          \c'
mvdwn 1
thisline 1
echo ' [Delete all files in directory?] \c'
mvdwn 1
thisline 1
echo ' [Confirm each while deleting?] \c'

mvup 2
inverse
scrn
read dirfile
resetscr

inverse
scrn
read del
resetscr

inverse
scrn
read confirm
resetscr
thisline 1

# valid - sets flag to indicate valid input
# 0 - no          1 - yes
```

Feb 10 03:33 1987 remove Page 2

```
valid=0

# if del is null, user chose default
if [ -z "$del" ]
then
    valid=1
    del=0
fi

# does user want to delete all files?
while [ "$valid" -eq 0 ]
do
    case $del in

        # user wants to delete all files
        y | Y | yes | Yes ) valid=1
                           del=1 ;;

        # user does not want to delete all files
        n | N | no | No ) valid=1
                       del=0 ;;

        # invalid answer
        * ) echo 'Delete all files response invalid, delete all files ok? \c '
           read del

        # user entered carriage return, default value
        if [ -z "$del" ]
        then
            valid=1
            del=0
        fi

        # user entered something else, loop back up to case to check it
    esac
done

# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0

# if confirm is null, user entered default
if [ -z "$confirm" ]
then
    valid=1
    confirm=0
fi

# does user want to confirm each?
while [ "$valid" -eq 0 ]
do
    case $confirm in

        # user wants to confirm each
        y | Y | yes | Yes ) valid=1
    esac
done
```

Feb 10 03:33 1987 remove Page 3

```
confirm=1 ;;

# user does not want to confirm each
n : N : no : No ) valid=1
confirm=0 ;;

# invalid answer
* ) echo $confirm 'is neither Yes nor No, confirm each? \c '
read confirm

# user entered carriage return, default value
if [ -z "$confirm" ]
then
    valid=1
    confirm=0
fi

# user entered something else, loop back up to case to check it
esac
done

# a directory? no
if [ ! -d $dirfile ]
then
    echo $dirfile: no such directory - can not remove it.
    exit
fi

# exists

# if user does not want to delete all files in directory, then
# can not remove the directory if it has files
if [ "$del" = 0 ]
then

# determine if directory is empty by listing its contents and
# piping that to wc to count the number of characters
count=`ls $dirfile | wc -c`
# if count is 0, then directory is empty and can be removed
if [ "$count" = 0 ]
then
    rmdir $dirfile
    echo Removing directory $dirfile ... done
# count not zero - files in directory, can not remove it
else
    echo Directory $dirfile is not empty, can not remove it.
fi
exit
fi

# set flag to show no subdirectories
somedir=0
```

Feb 10 03:33 1987 remove Page 4

```
files=`ls $dirfile`
for file in $files
do
    if [ -d $dirfile/$file ]
    then
        somedir=1
    fi
done

if [ "$somedir" -eq 1 ]
then
    echo Directory contains subdirectories.
    echo Can not delete $dirfile until subdirectories are removed.
    exit
fi

# no subdirectories - can delete

# user indicated wanted to confirm each, one at a time
if [ "$confirm" -eq 1 ]
then
    # set flag to 1 to show ok to delete directory
    # change it to 0 if user does not want to delete a file
    # therefore cannot delete directory
    okflg=1
    for file in $files
    do
        echo "Delete $file (y or n) ? \c"
        valid=0
        while [ "$valid" -eq 0 ]
        do
            read check
            echo
            case $check in
                #
                ok to delete
                y) rm $dirfile/$file
                   echo Deleting $file ... done.
                   valid=1 ;;
                #
                don't delete
                n ) valid=1
                   okflg=0 ;;
                #
                invalid response
                * ) echo $response 'is neither Yes nor No, reenter \c ' ;;
            esac
        done
    done

    # user does not want to confirm each, so delete each in turn
    else
        okflg=1
        for file in $files
        do
            rm $dirfile/$file
```

Feb 10 03:33 1987 remove Page 5

```
        echo Deleting $file ... done.
    done
fi
if [ "$okflg" -eq 1 ]
then
    rmdir $dirfile
    echo Removing directory $dirfile ... done.
else
    echo Directory $dirfile is not empty, can not remove it.
fi
)
# end rmdprmt

scrn () {
# resets the rectangle for reading input

thisline 36
rectangle 1 40

}
# end scrn

resetscr () {
normal
mvup
scrn
mvdwn 1
}
# end resetscr

remdir () {
# called by maindriver

# is it a directory?
if [ ! -d $name ]
then
    echo $name: no such directory - can not remove it.
    exit
fi

# determine if directory is empty by listing its contents and
# piping that to wc to count the number of characters

count=`ls $name | wc -c`

# if count is 0, then directory is empty and can be removed
if [ "$count" -eq 0 ]
then
    rmdir $name
    echo Removing directory $name ... done.

# count not zero - files in directory, can not remove it
else
    echo Directory $name is not empty, can not remove it.
fi
fi
```

Feb 10 03:33 1987 remove Page 6

```
)
# end remdir

# remove directory - MAIN DRIVER

# test for missing argument
# if argument missing, assume user wants to be prompted for input
if [ $# -eq 0 ]
then
    echo Did not specify directory - reenter
    exit
fi

# if number of arguments is more than 2, invalid use of command
if [ $# -gt 2 ]
then
    echo Too many arguments - reenter command
    exit
fi

# if first argument is "directory" - user wants to remove a directory
if [ "$1" = directory ]
then

    # if only 2 arguments - user does not want prompting
    if [ $# -eq 2 ]
    then
        name=$2
        remdir
        exit
    fi

    # user wants prompting
    else
        rmdprmt
        exit
    fi
fi

# invalid use of remove command
echo No such command - Remove $1 - reenter command.
```

Jan 5 17:08 1987 rename Page 1

```
# *****
# shell script name : rename
# November 28, 1986
# R. Loffman

# this shell uses "mv" to imitate BTOS' "rename"

# this uses cursor control routines written in C -
#     inverse
#     mvdwn
#     mvup
#     normal
#     rectangle
#     thisline

# *****
renprmt () {
# renprmt is like BTOS, i.e., it prompts the user for input

# trap the user hitting delete key - don't want to remain in inverse

trap "normal; mvdwn 4; exit" 2

mvdwn
echo Rename

mvdwn 1
echo ' Old file name      \c'
mvdwn 1
thisline 1
echo ' New file name      \c'
mvdwn 1
thisline 1
echo ' [Overwrite ok?]     \c'
mvdwn 1
thisline 1
echo ' [Confirm each?]    \c'

mvup 3
inverse
scrn
read oldfile
resetscr

inverse
scrn
read newfile
resetscr

inverse
scrn
read over
resetscr

inverse
scrn
```

Jan 5 17:08 1987 rename Page 2

```
read confirm
resetscr
```

```
thisline 1
```

```
# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0
```

```
# if over is null, user chose default
if [ -z "$over" ]
then
    valid=1
    over=0
fi
```

```
# does user want to overwrite?
while [ "$valid" -eq 0 ]
do
    case $over in
```

```
# user wants to overwrite
    y : Y : yes : Yes ) valid=1
                        over=1 ;;
```

```
# user does not want to overwrite
    n : N : no : No ) valid=1
                     over=0 ;;
```

```
# invalid answer
    * ) echo 'Overwrite response invalid, overwrite ok? \c '
        read over
```

```
# user entered carriage return, default value
    if [ -z "$over" ]
    then
        valid=1
        over=0
    fi
```

```
# user entered something else, loop back up to case to check it
```

```
    esac
done
```

```
# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0
```

```
# if confirm is null, user chose default
if [ -z "$confirm" ]
then
    valid=1
    confirm=0
fi
```

Jan 5 17:08 1987 rename Page 3

```
# does user want to confirm each?
while [ "$valid" -eq 0 ]
do
    case $confirm in

# user wants to confirm each
    y | Y | yes | Yes ) valid=1
                        confirm=1 ;;

# user does not want to confirm each
    n | N | no | No ) valid=1
                     confirm=0 ;;

# invalid answer
    * ) echo 'Confirm response invalid, confirm each? \c '
        read confirm

# user entered carriage return, default value
    if [ -z "$confirm" ]
    then
        valid=1
        confirm=0
    fi

    esac
done

# is the file to be renamed a directory? yes
if [ -d $oldfile ]
then
    echo $oldfile is a directory ... can not rename.
    exit
fi

# exist? no
if [ ! -s $oldfile ]
then
    echo $oldfile: no such file ... can not rename.
    exit
fi

# is second file a directory?
# yes
if [ -d $newfile ]
then
    echo Did not rename $oldfile, $newfile a directory.
    exit
fi

# does newfile exist? no
if [ ! -s $newfile ]
then

# does user want to confirm? yes
    if [ "$confirm" -eq 1 ]
```

Jan 5 17:08 1987 rename Page 4

```
then
    echo "Rename soldfile to $newfile? \c"
    valid=0
    while [ "$valid" -eq 0 ]
    do
        read check
        echo
        case $check in
            #      ok to rename
              y) prnover
                 valid=1 ;;
            #      don't rename
              n) echo Did not rename $newfile to soldfile.
                 valid=1 ;;
            #      invalid response
              *) echo $response 'is neither Yes nor No, reenter \c ' ;;
              esac
        done

    # user doesn't want to confirm
    else
        prnover
    fi

    # newfile exists
    else

    # does user want to confirm each? yes
    if [ "$confirm" -eq 1 ]
    then

        echo "Rename soldfile to $newfile? \c"
        valid=0
        while [ "$valid" -eq 0 ]
        do
            read check
            echo
            case $check in
                #      ok to rename
                  y) if [ "$tover" -eq 1 ]
                      then
                          prover
                          valid=1
                      else
                          echo Did not rename soldfile, $newfile already exists
                          valid=1
                      fi ;;
            #      don't rename
                  n) echo Did not rename $newfile to soldfile.
                     valid=1 ;;
            #      invalid response
                  *) echo $response 'is neither Yes nor No, reenter \c ' ;;
                  esac
        done

    # user doesn't want to confirm
    else
        prnover
    fi
fi
```

Jan 5 17:08 1987 rename Page 5

```
#         invalid response
# ) echo $response 'is neither Yes nor No, reenter \c ' ;;
esac
done

# user doesn't want to confirm
else

# does user want to overwrite? yes
if [ "$over" -eq 1 ]
then
    prover
fi

# user doesn't want to overwrite
else
    echo Did not rename $oldfile, $newfile already exists
fi
fi

# end big if
fi
)
# end renprmt

scrn () {
# resets the rectangle for reading input

thisline 21
rectangle 1 50

}
# end scrn

resetscr () {

normal
mvup
scrn
mvdwn 1
}
# end resetscr

prnover () {
# called from renprmt

mv $oldfile $newfile
echo Renaming $oldfile to $newfile ... done.
}
# end prnover

prover () {
# called from renprmt

mv $oldfile $newfile
```

Jan 5 17:08 1987 rename Page 6

```
echo Renaming $oldfile to $newfile \((overwriting\) ... done.
)
# end prover
```

```
# rename - MAIN DRIVER
```

```
# test for missing argument
# if argument missing, assume user wants to be prompted for input
if [ $# -eq 0 ]
then
    renprmt
    exit
fi
```

```
# if user supplies argument(s), assume user does not want to be
# prompted for input.
# validate user supplied input
```

```
# one argument - not enough
if [ $# -eq 1 ]
then
    echo Not enough files listed - reenter.
    exit
fi
```

```
# more than two arguments - too many
if [ $# -gt 2 ]
then
    echo More than two files listed - reenter.
    exit
fi
```

```
# first file a directory?
```

```
# yes - can not do rename
if [ -d $1 ]
then
    echo $1 is directory - can not rename.
    exit
fi
```

```
# does first file exist?
```

```
# no - rename with it does not make sense
if [ ! -s $1 ]
then
    echo $1: no such file - can not rename.
    exit
fi
```

```
# first file exists
```

```
# is second file a directory?
# yes
```

Jan 5 17:08 1987 rename Page 7

```
if [ -d $2 ]
then
    echo Did not rename $1, $2 a directory.
    exit

# does the second file exist?
# no - create it
elif [ ! -s $2 ]
then
    mv $1 $2
    echo Renaming $1 to $2 ... done.
    exit

# second file exists, overwrite
else
    echo Did not rename $oldfile, $newfile already exists.

fi
```

Feb 9 16:20 1987 set Page 1

```
# *****
# shell script name : set
# November 29, 1986
# R. Loffman

# this shell uses "chmod" to imitate BTOS' "set protection" and
# "set directory protection"

# because there is already a system command set, the system command
# will be the one that is executed instead of this version of set.

# to use this version of set, either
# 1. specify full path name to this set or
# 2. make the path of the user include the path to the
#    directory where this is located and place this path
#    before the path to the system's set. THIS IS NOT ADVISED

# this uses cursor control routines written in C -
#     inverse
#     mvdown
#     mvup
#     normal
#     rectangle
#     thisline

# *****

setpmt () {
# called by main driver

# trap the user hitting the delete key - don't want to be left in
# inverse mode

trap "normal; mvdown 4; exit" 2

# check if is file or directory set protection
# kind was set in main driver

if [ "$kind" = f ]
then
    mvdown
    echo Set protection
    mvdown 1
    echo '   File list                \c'
else
    echo Set directory protection
    mvdown 1
    echo '   Directory name            \c'
fi

mvdown 1
thisline 1
echo '   New protection level '\(15, 5 or 0\) ' \c'

mvdown 1
```

Feb 9 16:20 1987 set Page 2

```
thisline 1
echo ' [Confirm each?] \c'

mvup 2
inverse
scrn
read files
resetscr

inverse
scrn
read prot
resetscr

inverse
scrn
read confirm
resetscr

thisline

# check for valid protection level
ckprot

# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0

# if confirm is null, user chose default
if [ -z "$confirm" ]
then
    valid=1
    confirm=0
fi

# does user want to confirm each?
while [ "$valid" -eq 0 ]
do
    case $confirm in

# user wants to confirm each
    y | Y | yes | Yes ) valid=1
                        confirm=1 ;;

# user does not want to confirm each
    n | N | no | No ) valid=1
                     confirm=0 ;;

# invalid answer
    * ) echo $confirm 'is neither Yes nor No, confirm each? \c '
        mvup 1
        inverse
        scrn
        read confirm
        resetscr
    esac
done
```

Feb 9 16:20 1987 set Page 3

```
thisline 1

# user entered carriage return, default value
if [ -z "$confirm" ]
then
    valid=1
    confirm=0
fi

# user entered something else, loop back up to case to check it

esac
done

mvdwn 3

# goflg indicates ok to go ahead and set protection
# set for each file
# 1 - ok          0 - not ok

# read files to be protected
for file in $files
do

# initialize
goflg=1

# a directory? yes
if [ -d $file ]
then
    if [ "$kind" = f ]
    then
        echo $file is a directory - can not set protection as a file.
        goflg=0
    fi

    elif [ "$kind" = f ]
    then

# does the file exist? no
if [ ! -s $file ]
then
    echo $file does not exist - can not set protection.
    goflg=0
fi

else [ "$kind" = d ]
if [ ! -d $file ]
then
    echo $file ' is not a directory - can not set \c'
    echo protection as a directory.
    goflg=0
fi
fi

# check goflg to see if file/directory was valid for type of set
```

Feb 9 16:20 1987 set Page 4

```
if [ "$gofig" -eq 1 ]
then
# user indicated wanted to confirm each, one at a time
if [ "$confirm" -eq 1 ]
then
    echo "Set protection for $file (y or n)? \c"
    valid=0
    while [ "$valid" -eq 0 ]
    do
        read check
        echo
        case $check in
            #      ok to set protection
            y | yes | Y | Yes ) chprot
                                valid=1 ;;

            #      don't set protection
            n | no | N | No ) valid=1 ;;

            #      invalid response
            * ) echo $response 'is neither Yes nor No, reenter \c ' ;;
        esac
    done

# user does not want to confirm each, so set protection for
# each in turn
else
    chprot
fi
done
)

# end of setpmt

scrn () {
# resets the rectangle for reading input

thisline 37
rectangle 1 30

}
# end scrn

resetscr () {
normal
mvup
scrn
mvdwn 1

}
# end resetscr
```

Feb 9 16:20 1987 set Page 5

```
chprot () {
# called by main driver and setpmt

chmod $prot $file
echo Setting protection for $file ... done.
}
# end chprot

ckprot () {
# called by main driver and setpmt
# check that protection is valid
if [ "$prot" = 15 ]
then
    prot=777
elif [ "$prot" = 5 ]
then
    prot=555
elif [ "$prot" = 0 ]
then
    prot=700
else
    echo $prot is an invalid protection level.
    exit
fi
}
# end ckprot

# set - MAIN DRIVER

# if argument missing
if [ $# -eq 0 ]
then
    echo Did not specify protection.
    exit
fi

# if first argument is "protection" - user wants to set protections
# on a file
if [ "$1" = protection ]
then
    # if only 1 argument - user wants prompting
    if [ $# -eq 1 ]
    then
        kind=f
        setpmt
        exit
    # user does not want prompting
    else

        # save protection field to send to ckprot
        # because calling another function will wipe out $ arguments
        # shift all files into holding variable so not to lose them

        prot=$2
```

Feb 9 16:20 1987 set Page 6

```
    shift 2
    files=
    while [ "$#" -ne 0 ]
    do
        files="$files $1"
        shift
    done
    ckprot

#    set protection for each file

    for file in $files
    do

#        is file a directory?
#        yes - then can not set protection as a file
        if [ -d $file ]
        then
            echo $file ' is a directory name - can not set \c'
            echo protection as a file.

#            does the file exist?
#            no
            elif [ ! -s $file ]
            then
                echo File $file: no such file.

#            file exists
            else
                chprot
            fi

        done
        exit

    fi
    exit
fi

# user is setting directory protection
if [ "$1" = directory ]
then
    if [ "$2" = protection ]
    then
        if [ $# -eq 2 ]
        # user wants prompting
        then
            kind=d
            setpmt
            exit

# not enough arguments
        elif [ $# -eq 3 ]
        then
            echo Not enough information given.
```

Feb 9 16:20 1987 set Page 7

```
        exit

#   user does not want prompting
else

#       save protection field to send to ckprot
#       because calling another function will wipe out $ arguments
#       shift all files into holding variable so not to lose them

    prot=$3

    shift 3
    files=
    while [ "$#" -ne 0 ]
    do
        files="$files $1"
        shift
    done
    ckprot

#       set protection for each file

    for file in $files
    do

#           is file a directory?
#           no - then can not set protection as a directory
#           if [ ! -d $file ]
#           then
#               echo $file 'is not a directory name - can not \c'
#               echo set protection as a directory.
#           directory exists
#           else
#               chprot
#           fi

#           get next directory
        done
        exit
    fi
fi

# invalid use of set command
echo No such command - set $1 - reenter command.
```

```
# *****
# shell script name: sort
# December 12, 1986
# R. Loffman

# this shell uses "sort" to imitate BTOS' "sort"

# this shell uses cursor control routines written in C -
#     inverse
#     mvdwn
#     mvup
#     normal
#     rectangle
#     thisline

# because there is already a system command sort, the system command
# will be the one that is executed instead of this version of sort.

# to use this version of sort, either
#     1. specify full path name to this sort or
#     2. make the path of the user include the path-to the
#         directory where this is located and place this path
#         before the path to the system's sort. THIS IS NOT ADVISED
# *****
sortprmt () {
# sortprmt is like BTOS, i.e., it prompts the user for input

# trap the user hitting the delete key - don't want to be left in
# inverse mode

trap "normal; mvdwn 4; exit" 2

mvdwn
echo Sort

mvdwn
echo '  Input files          \c'

mvdwn 1
thisline 1
echo '  Output file          \c'

mvdwn 1
thisline 1
echo '  Beginning sort position \c'

mvdwn 1
thisline 1
echo '  Ending sort position   \c'

mvup 3
inverse
scrn
read files
resetscr
```

```
inverse
scrn
read outfile
resetscr
```

```
inverse
scrn
read beginpos
resetscr
```

```
inverse
scrn
read endpos
resetscr
```

```
thisline 1
```

```
# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0
```

```
# check if user entered valid number for beginpos
# done by checking if 1 > input ie., input is not a number
if [ 1 -gt "$beginpos" ]
then
  while [ "$valid" -eq 0 ]
  do
```

```
    # invalid answer
    echo 'Beginning sort position not a number - reenter \c'

    read beginpos

    if [ 1 -gt "$beginpos" ]
    then
      valid=0
    else
      valid=1
    fi

  done
fi
```

```
# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0
```

```
# check if user entered valid number for endpos
# done by checking if 1 > input ie., input is not a number
if [ 1 -gt "$endpos" ]
then
  while [ "$valid" -eq 0 ]
  do
```

```
    # invalid answer
    echo 'Ending sort position not a number - reenter \c'
```

```
    read endpos

    if [ 1 -gt "$endpos" ]
    then
        valid=0
    else
        valid=1
    fi

done
fi

# build list of files to sort
sortfiles=

# no files to start with
somefiles=0

# no files to sort yet
flg=0

# read files to be sorted
for file in $files
do

    # file a directory?
    # yes - don't sort it
    if [ -d $file ]
    then
        echo $file is directory, can not sort it.

    # file exist?
    # no - sort doesn't make sense
    elif [ ! -s $file ]
    then
        echo $file does not exist, can not sort it.

    # ok to sort
    elif [ "$flg" -eq 0 ]
    then
        flg=1
        somefiles=1
        sortfiles="$sortfiles $file"

    # ok to sort and some already there
    else
        sortfiles="$sortfiles $file"
    fi

# get next file in list
done

# nothing to Sort
if [ "$somefiles" -eq 0 ]
then echo All of the files were directories or did not exist.
```

Feb 9 16:24 1987 sort Page 4

```
        echo Sort did not occur.
        exit
    fi

# check file to put results into
# file a directory?
# yes - can't sort into it
if [ -d $outfile ]
then
    echo $outfile is directory, can not sort into it.
    exit
fi

# not a directory

# note- will overwrite output file if it exists
if [ -s $outfile ]
then
    echo Sorting files into $outfile \ (overwriting\ ) ... done.
else
    echo Sorting files into $outfile ... done.
fi

/bin/sort -o$outfile +$beginpos -tendpos $sortfiles
)
# end sortprmt

scrn () {
# resets rectangle for reading input

thisline 28
rectangle 1 50

}
# end scrn

resetscr () {

normal
mvup
scrn
mvdwn 1

}
# end resetscr

# sort - MAIN DRIVER

# test for missing argument
# if argument missing, assume user wants to be prompted for input
if [ $# -eq 0 ]
then
    sortprmt
    exit
fi
```

```
fi

# must supply 4 arguments - input files, output file,
# beginning position and ending positions

# did not supply enough arguments
if [ $# -eq 1 ]
then
    echo Command requires a minimum of four arguments:
    echo begin-sort-position end-sort-positiona input-file output-file
    echo Reenter command.
    echo
    exit
fi

# did not supply enough arguments
if [ $# -eq 2 ]
then
    echo Command requires a minimum of four arguments:
    echo begin-sort-position end-sort-positiona input-file output-file
    echo Reenter command.
    echo
    exit
fi

# did not supply enough arguments
if [ $# -eq 3 ]
then
    echo Command requires a minimum of four arguments:
    echo begin-sort-position end-sort-positiona input-file output-file
    echo Reenter command.
    echo
    exit
fi

# did not supply enough arguments
if [ $# -eq 4 ]
then
    echo Command requires a minimum of four arguments:
    echo begin-sort-position end-sort-positiona input-file output-file
    echo Reenter command.
    echo
    exit
fi

# if user supplies argument(s), assume user does not want to be
# prompted for input.
# validate user supplied input

beginpos=$1
endpos=$2

# check if user entered valid number for beginpos
# done by checking if 1 > input ie., input is not a number
if [ 1 -gt "$beginpos" ]
then
```

```
    echo Beginning sort position not a number - reenter command.
    exit
fi

# check if user entered valid number for endpos
# done by checking if 1 > input ie., input is not a number
if [ 1 -gt "$endpos" ]
then
    echo Ending sort position not a number - reenter command
    exit
fi

# user supplied enough arguments, need to check validity of
# selected files - shift to them

shift 2

# build list of files to sort
sortfiles=

# no files to start with
somefiles=0

# no files to sort yet
flg=0
while [ "$#" -ne 1 ]
do

    # file a directory?
    # yes - don't sort it
    if [ -d $1 ]
    then
        echo $1 is directory, can not sort it.

    # file exist?
    # no - sort doesn't make sense
    elif [ ! -s $1 ]
    then
        echo $1 does not exist, can not sort it.

    # ok to sort
    elif [ "$flg" -eq 0 ]
    then
        flg=1
        somefiles=1
        sortfiles="$sortfiles $1"

    # ok to sort and already some there
    else
        sortfiles="$sortfiles $1"
    fi

    # get next file in list
    shift
done
```

Feb 9 16:24 1987 sort Page 7

```
# nothing to Sort
if [ "$somefiles" -eq 0 ]
then echo All of the files were directories or did not exist.
    echo Sort did not occur.
    exit
fi

# last file is file to put results into
# file a directory?
# yes - can't sort into it
if [ -d $1 ]
then
    echo $1 is directory, can not sort into it.
    exit
fi

# not a directory
outfile=$1

# note- will overwrite output file if it exists
if [ -s "$outfile" ]
then
    echo Sorting files into $outfile \((overwriting\) ... done.
else
    echo Sorting files into $outfile ... done.
fi

/bin/sort -o$outfile +$beginpos -tendpos $sortfiles
```

Feb 9 16:32 1987 type Page 1

```
# *****
# shell script name : type
# November 28, 1986
# R. Loffman

# this shell uses "more" to imitate BTOS' "type"

# the BTOS type displays one screenful at a time to the terminal. The
# user presses the next command to continue to the next screen or
# presses the cancel key to stop.

# in the Centix version which users "more", the user presses the space
# bar to get the next screenful. Pressing the delete key will
# completely interrupt the execution of the command. This means that
# if there are several files to be typed and the user presses the
# delete key during the displaying of the first file, the whole process
# terminates

# because there is already a system command type, the system command
# will be the one that is executed instead of this version of type.

# to use this version of type, either
# 1. specify the full path name to this type or
# 2. make the path of the user include the path to the
#    directory where this is located and place this path
#    before the path to the system's type. THIS IS NOT ADVISED

# this uses cursor control routines written in C -
#     inverse
#     mvdwn
#     mvup
#     normal
#     rectangle
#     thisline

# *****

# typprmt is like BTOS, i e, it prompts the user for input

typprmt () {

# trap the user hitting the delete key - don't want to end up in
# inverse mode

trap "normal; mvdwn 4; exit" 2

mvdwn
echo Type

mvdwn
echo ' File list          \c'
mvdwn 1
thisline 1
echo ' [Confirm each?]    \c'

mvup 1
```

Feb 9 16:32 1987 type Page 2

```
inverse
scrn
read files
resetscr

inverse
scrn
read confirm
resetscr

thisline 1

# valid - sets flag to indicate valid input
# 0 - no          1 - yes
valid=0

# if confirm is null, user entered default
if [ -z "$confirm" ]
then
    valid=1
    confirm=0
fi

# does user want to confirm each ?
while [ "$valid" -eq 0 ]
do
    case $confirm in
        # user wants to confirm each
        y | Y | yes | Yes ) valid=1
                           confirm=1 ;;

        # user does not want to confirm each
        n | N | no | No ) valid=1
                        confirm=0 ;;

        # invalid answer
        * ) echo $confirm 'is neither Yes nor No, confirm each? \c '
           mvup 1
           inverse
           scrn
           read confirm
           resetscr
           thisline 1

        # user entered carriage return, default value
        if [ -z "$confirm" ]
        then
            valid=1
            confirm=0
        fi

        # user entered something else, loop back up to case to check it
    esac
done
```

Feb 9 16:32 1987 type Page 3

done

mvdown 3

```
# read files to be typed
for file in $files
do
# a directory? yes
if [ -d $file ]
then
echo
echo $file is a directory - can not type it.

# exist? no
elif [ ! -s $file ]
then
echo
echo $file:
echo $file: No such file.

# exists

# user indicated wanted to confirm each, one at a time
elif [ "$confirm" -eq 1 ]
then
echo "Type $file (y or n) ? \c"
valid=0
while [ "$valid" -eq 0 ]
do
read check
echo
case $check in

#      ok to type
y) echo $file:
more $file
echo Typed $file.
valid=1 ;;

#      don't type
n ) valid=1 ;;

#      invalid response
* ) echo $response 'is neither Yes nor No, reenter \c ' ;;
esac
done

# user does not want to confirm each, so type each in turn
else
echo
echo $file:
more $file
echo Typed $file.
fi
done
)
```

Feb 9 16:32 1987 type Page 4

```
# end typprompt

scrn () {
# resets rectangle for reading input

thisline 21
rectangle 1 50

}
# end scrn

resetscr () {

normal
mvup
scrn
mvdwn 1

}
# end resetscr

# type - MAIN DRIVER

# test for missing argument
# if argument missing, assume user wants to be prompted for input
if [ $# -eq 0 ]
then
    typprompt
    exit
fi

# if user supplies argument(s), assume user does not want to be
# prompted for input.
# validate user supplied input

# need to shift thru all of them

while [ "$#" -ne 0 ]
do

#   file a directory?
#   yes - don't type it
    if [ -d $1 ]
    then
        echo $1 is a directory, can not type it.

#   file exist?
#   no - type doesn't make sense
    elif [ ! -s $1 ]
    then
        echo $1: No such file.

#   ok to type
    else
```

Feb 9 16:32 1987 type Page 5

```
    echo $1  
    more $1  
    echo Typed $1.  
fi  
  
# get next file in list  
  shift  
done
```

APPENDIX H

```
# shell script name: unixhelp
# R. Loffman
# November 22, 1986
# this is a program to give users help wit
```

```
detail () {
# called by main driver
```

```
# clear screen
```

```
clear
```

```
echo      This is a list of available commands
echo      You may select one for additional i
echo      Enter the first word for commands v
```

```
echo
echo      'command'                                li
echo      -----
echo      append                                     ty
echo      'cat'                                       ct
echo      'copy'                                      st
echo      'chmod'                                    st
echo      'cp'                                        ct
echo      'create' \((file\)
echo      'create' \((directory\)
echo      'delete'                                   ri
echo      'ed'                                       ed
echo      'edit'                                     ed
echo      'files'                                    li
echo      'ls'                                       fi
echo      'print'                                    pr
echo      'remove' \((directory\)
echo      'rename'                                   m
echo      'rm'                                       d
echo      'rmdir'                                    r
echo      'set' \((directory protection\)
echo      'set' \((protection\)
echo      'sort'                                     c
echo      'type'                                     c
echo      'vi'                                       e
```

```
echo      ' Enter a command name: \c'
```

```
read name
```

```
echo
```

```
echo
```

```
case $name in
```

```
    append ) app ;;
```

```
    cat ) typ2 ;;
```

```
    chmod ) set2 ;;
```

```
    copy ) copy1 ;;
```

```
    cp ) copy2 ;;
```

```
    create ) cr ;;
```

```
    delete ) del1 ;;
```

```
    ed ) ed1 ;;
```

```
    edit ) ed2 ;;
```

```
    files ) ls1 ;;
```

```
    ls ) ls2 ;;
```

```
mv ) ren2 ;;
print ) pr1 ;;
remove ) rem1 ;;
rename ) ren1 ;;
rm ) del2 ;;
rmdir ) rem2 ;;
set ) set1 ;;
sort ) sort ;;
type ) typ1 ;;
vi ) ed3 ;;
*) echo 'Invalid choice' ;;

esac

)
# end detail

app () {
echo 'append is used to append files together and place them'
echo '    into one file.'
form
echo '    append from-file\ (s) to-file '
prmt
}

copy1 () {
echo 'copy is used to make a copy of a file.'
care
echo 'Can overwrite existing file.'
form
echo '    copy from-file to-file '
prmt
}

copy2 () {
echo 'cp is used to make a copy of a file.'
care
echo 'Can overwrite existing file.'
form
echo '    cp from-file to-file '
prmt
}

cr () {
echo 'create is used to create a file or a directory.'
form
echo '    create file file-name '
echo '    or'
echo '    create directory directory-name.'
prmt
}

set1 () {
echo 'set is used to set protection for a file or a directory.'
form
echo '    set protection protection-level file-name '
echo '    or'
}
```

```
echo '    set directory protection protection-level directory-name.'
prmt
)
```

```
set2 () {
echo 'chmod is used to set protection for a file or a directory '
form
echo '    chmod protection-level file-name '
echo '    or'
echo '    chmod protection-level directory-name.'
prmt
)
```

```
del1 () {
echo 'delete is used to delete a file.'
care
form
echo '    delete file-name'
prmt
)
```

```
del2 () {
echo 'rm is used to remove a file.'
care
form
echo '    rm file-name'
prmt
)
```

```
typ1 () {
echo 'type is used to display the contents of a file to the screen.'
form
echo '    type file-name'
prmt
)
```

```
typ2 () {
echo 'cat is used to display the contents of a file to the screen.'
form
echo '    cat file-name'
prmt
)
```

```
ren1 () {
echo 'rename is used to rename a file.'
care
form
echo '    rename old-file-name new-file-name'
prmt
)
```

```
ren2 () {
echo 'mv is used to move a file to a new file-name.'
care
form
echo '    mv old-file-name new-file-name'
```

```
prmt
)
```

```
rem1 () {
echo 'remove directory is used to remove a directory.'
care
form
echo '    remove directory directory-name'
prmt
}
```

```
rem2 () {
echo 'rmdir is used to remove a directory.'
care
form
echo '    rmdir directory-name'
prmt
}
```

```
ed1 () {
echo 'ed is used to edit the contents of a file.'
form
echo '    ed file-name'
prmt
}
```

```
ed2 () {
echo 'edit is used to edit the contents of a file.'
form
echo '    edit file-name'
prmt
}
```

```
ed3 () {
echo 'vi is used to edit the contents of a file.'
form
echo '    vi file-name'
prmt
}
```

```
pr1 () {
echo 'print is used to print the contents of a file to a printer.'
form
echo '    print file-name'
prmt
}
```

```
ls1 () {
echo 'files is used to list file names.'
form
echo '    files file-name\(\s\)'
prmt
}
```

```
ls2 () {
echo 'ls is used to list file names.'
```

```
form
echo '    ls file-name\(\s\)'
echo 'More detail can be displayed by supplying options.'
echo 'For example '
echo '    ls -al file-name'
prmt
}

sort () {
echo 'sort is used to sort lines of files.'
care
echo 'Can overwrite existing file.'
form
echo '    sort begin-position end-position input-files output-file'
prmt
}

prmt () {
echo
echo 'If the user does not supply arguments he will be'
echo 'prompted for input as in BTOS.'
echo
}

form () {
echo
echo 'The form of the command is:'
}

care () {
echo 'Use this command with caution.'
}

# unixhelp - MAIN DRIVER

# test for missing argument

if [ $# -eq 0 ]
then
    detail
    exit
fi

# if number of arguments is more than one, invalid use of command
if [ $# -gt 1 ]
then
    echo 'Too many arguments - reenter command.'
    exit
fi
echo

if [ "$1" = append ]
then
    app
elif [ "$1" = cat ]
then
```

```
    typ2
elif [ "$1" = chmod ]
then
    set2
elif [ "$1" = copy ]
then
    copy1
elif [ "$1" = cp ]
then
    copy2
elif [ "$1" = create ]
then
    cre
elif [ "$1" = delete ]
then
    del
elif [ "$1" = ed ]
then
    ed1
elif [ "$1" = edit ]
then
    ed2
elif [ "$1" = files ]
then
    ls1
elif [ "$1" = ls ]
then
    ls2
elif [ "$1" = mv ]
then
    ren2
elif [ "$1" = print ]
then
    pr1
elif [ "$1" = remove ]
then
    rem1
elif [ "$1" = rename ]
then
    ren1
elif [ "$1" = rm ]
then
    del2
elif [ "$1" = rmdir ]
then
    rem2
elif [ "$1" = set ]
then
    set1
elif [ "$1" = sort ]
then
    sort
elif [ "$1" = type ]
then
    typ1
elif [ "$1" = vi ]
```

Dec 12 22:05 1986 unixhelp Page 7

```
then
    ed3
else
    echo $1 ' not in help facility.'
fi
```

APPENDIX I

```
# This script is /bin/candidate. It is the shell which cand1 logs into.
# The main program, which appears at the end of this file, consists of one
# line which is a call to mainmenu.
# mainmenu is the PERSONNEL OFFICE MAIN MENU.
#   screen name - ACP
#   purpose - provide menu to applications on the system
```

```
# The other subfunction scripts are
#   subfunction name   screen name   purpose
#   -----
#   help               ACPHLP        explains options from ACP
#
#   ksaitemanalysis    KSA030        analysis of KSAs, TASKs and
#                                     items
#
#   ksalink             KSA020        allows viewing of KSAs and
#                                     TASKs on file, and the adding,
#                                     editing, deleting or retrieval
#                                     of a linked item
#
#   ksamaint            KSA010        allows the adding, editing,
#                                     deleting or retrieval of KSAs
#                                     or TASKs
#
#   ksareports          KSARPT        allows the printing of reports
#                                     related to the item bank to the
#                                     terminal, parallel, or serial
#                                     printer
#
#   ksabank             KSA           the menu into the KSA ITEM BANK
#                                     (KSA010, KSA020, KSA030, KSARPT)
#
#   candmenu            CE           the menu into the CANDIDATE
#                                     EVALUATION SYSTEM. Currently,
#                                     only the KSA ITEM BANK is
#                                     implemented.
```

```
# see mainmenu for globally set variables.
```

```
help () {
```

```
# shell script: help
# R. Loffman
# October 1986
```

```
# explains options from Personnel Office Main Menu
```

```
# called by mainmenu
```

```
clear
```

```
echo '
```

```
*****
```

Feb 5 12.27 1987 candidate Page 2

```
*
*
*                                     (ACPHLP)
*
*      PERSONNEL OFFICE MAIN MENU
*      HELP FACILITY
*
*      0 will return you to the Centix login screen
*
*      1 will take you to the CANDIDATE EVALUATION SYSTEM MAIN MENU
*
*      2 is for demonstrating how other applications would be
*      accessed from this menu
*
*      0 and CODE-FINISH will take you to ETOS
*
*
* *****
*      Press return key when ready to continue \o'
* read keys
*
}
```

```
ksaitemanalysis () {
# shell script name: ksaitemanalysis
# R. Loffman

# test demo of item analysis screens

# calls from Ingres
#   analfrm1
#   analfrm2

# calls
#   ksareports
#   ksamaint
#   ksalink
#   candmenu

# Variable ingpth is exported from mainmenu. It is the path which
# describes the path to Ingres commands

# Variable clear clears the screen. It is exported from mainmenu

while true
do
    clear

    # display menu

echo '
*****
*
*                                     (KSA030)
*
*      KSA ANALYSIS
*
}
```

```

*                                     MAIN MENU
*
* 0 - Return to KSA ITEM BANK MAIN MENU (KSA)
*
* 1 - KSA Analysis
* 2 - Item Analysis
*
* r - return to calling menu
*
*****

Please enter selection and press RETURN \c'

# read and process selection

read choice
echo
case $choice in
#   user wants to return to KSA ITEM BANK MM
    0) ksabank ;;

#   user wants to review KSA Analysis
    1) echo 'This will take some time...'
      ${ingpth}qbf candeval analfrm1 -1 ;;

#   user wants to review Item Analysis
    2) echo 'This will take some time...'
      ${ingpth}qbf candeval analfrm2 -1 ;;

#   user wants to go (directly to KSA ITEM BANK MM
    ksa : KSA > ksabank ;;

#   user wants to go to KSA ITEM BANK MAINTENANCE MENU
    ksa010 : KSA010 > ksamaint ;;

#   user wants to go directly to KSA ITEM BANK LINKAGE MENU
    ksa020 : KSA020 > ksalink ;;

#   user wants to go directly to KSA ITEM BANK REPORTS MENU
    ksarpt : KSARPT > ksareports ;;

#   user wants to return to calling menu
    r > return ;;

#   user entered invalid choice
    *) echo $choice ' invalid entry' ;;
esac

done
)

```


Feb 5 12:27 1987 candidate Page 5

```
read choice
echo
case $choice in
#   user wants to return to KSA ITEM BANK MAIN MENU
    ksa : KSA : 0) ksabank ;;

#   user wants to see KSA numbers and titles
    1) echo 'This will take some time'
      $(ingpth)report candeval ksarpt1 ;;

#   user wants to see TASK numbers and definitions
    2) echo 'This will take some time'
      $(ingpth)report candeval tskrpt1 ;;

#   user wants to add, edit, delete, retrieve an item
    3) echo 'This will take some time'
      $(ingpth)qbf candeval lnkfrm1 -1 ;;

#   user wants to go directly to KSA ITEM BANK MAINTENANCE MENU
    ksa010 : KSA010 ) ksamaint ;;

#   user wants to go directly to KSA ITEM BANK ANALYSIS MENU
    ksa030 : KSA020 ) ksaitemanalysis ;;

#   user wants to go directly to KSA ITEM BANK REPORTS MENU
    ksarpt : KSARPT ) ksareports ;;

#   user wants to return to calling menu
    r ) return ;;

#   user entered invalid choice
    *) echo $choice ' invalid entry' ;;
esac

done
)
```

```
ksamaint () {

# shell script name: ksamaint
# R. Loffman
# October 1986

# this is the menu for the user to add, delete, change, retrieve
# elements from ksatable, tasktable

# calls from Ingres:
#   ksafirm1
#   taskfirm1

# calls
#   ksabank
```

Feb 5 12 27 1987 candidate Page 6

```
#      ksalink
#      ksaitemanalysis
#      ksareports
#      candmenu

# Variable ingpth is exported from mainmenu  It is the path
# to the Ingres commands

# Variable clear is exported from mainmenu.  It clears the screen.

while true
do
    clear

    # display menu

echo '
*****
*
*                                     (KSA010) *
*                                     *
*          KSA ITEM BANK              *
*          MAINTENANCE MENU           *
*                                     *
*  0 - Return to KSA ITEM BANK MAIN MENU (KSA) *
*                                     *
*  1 - Add, Edit, Delete, Retrieve KSA *
*  2 - Add, Edit, Delete, Retrieve TASK *
*                                     *
*  r - return to calling menu         *
*                                     *
*****

    Please enter selection and press RETURN  \c'

#      read and process selection

    read choice
    echo
    case $choice in
#      user wants to return to KSA ITEM BANK MAIN MENU
        ksa : KSA : 0) ksabank ;;

#      user wants to add, edit, delete, retrieve from ksatable
        1) echo 'This will take some time...'
           ${ingpth}qbf candeval ksafirm1 -1 ;;

#      user wants to add, edit, delete, retrieve from tasktable
        2) echo 'This will take some time...'
           ${ingpth}qbf candeval tsfirm1 -1 ;;

#      user wants to go directly to KSA ITEM BANK LINKAGE MENU
        ksa020 : KSA020 ) ksalink ;;

#      user wants to go directly to KSA ITEM BANK ANALYSIS MENU
        ksa030 : KSA030 ) ksaitemanalysis ;;
    esac
done
```

Feb 5 12:27 1987 candidate Page 7

```
#      user wants to go directly to KSA ITEM BANK REPORTS MENU
ksarpt : KSARPT ) ksareports ;;

#      user wants to return to calling menu
r ) return ;;

#      user entered invalid choice
*) echo $choice ' invalid entry' ;;
esac

done
)
```

ksareports () {

```
# shell script name: ksareports
# R. Loffman
# October 1986
```

```
# this is the menu for the user to print reports
# from ksatable, tasktable, linktable
```

```
# the user has the choice of printing
# 1. to the terminal
# 2. draft copy to SPL (parallel printer) using print
# 3. final copy to SPLB (serial printer) using lpr
```

```
# calls from Ingres.
# ksarpt2 - Print KSAs and definitions
# tskrpt1 - Print TASKS and definitions
# lnkrpt1 - Print linked item
```

```
# calls:
# candmenu
# ksamaint
# ksalink
# ksaitemanalysis
```

```
# Variable ingpth is exported from mainmenu. It is the path
# to the Ingres commands.
```

```
# Variable clear is exported from mainmenu. It clears the screen.
```

```
while true
do
```

```
    clear
```

```
    # display menu
```

```
echo '
*****
```

[illegible]

Please enter selection and press RETURN

```
# read and process selection

read choice
echo
# Check for user's report choice
case $choice in
#   user wants to return to KSA ITEM BANK MAIN MENU
    ksa : KSA : 0) ksabank
        ok=no ;;

#   user wants to review KSAs and titles
    1) rpt=ksarpt2 ;;

#   user wants to review TASKs and definitions
    2) rpt=tskrpt1 ;;

#   user wants to review linked items
    3) rpt=lnkrpt1 ;;

#   user wants to go directly to KSA ITEM BANK MAINTENANCE MM
    ksa010 : KSA010 ) ksamaint
        ok=no ;;

#   user wants to go directly to KSA ITEM BANK LINKAGE MENU
    ksa020 : KSA020 ) ksalink
        ok=no ;;

#   user wants to go directly to KSA ITEM BANK ANALYSIS MENU
    ksa030 : KSA030 ) ksaitemanalysis
        ok=no ;;

#   user wants to return to calling menu
    r ) return
        ok=no ;;

#   user made invalid selection
    *) echo $choice ' invalid entry '
        ok=no ;;

esac
```

Feb 5 12:27 1987 candidate Page 9

```
# print only if valid selection made
# give user choice of where to print
if [ "$ok" != no ]
then
    echo 'Do you want to print '
    echo '    - to the terminal(t)?'
    echo '    - draft copy (d)?'
    echo '    - final copy (f)?'
    echo 'Print choice (t d or f)? 'c'
    read prchoice
    echo
    case $prchoice in
#       t, T or word starting with t or T are valid
        [tT]* ) echo 'This will take some time...'
                $(ingpth)report candeval $rpt ;;

#       d, D, or word starting with D or D are valid
#       runs report, puts output in temporary directory, prints
#       from temporary directory to parallel printer
#       $$ - associates the report with the logged in process
        [dD]* ) echo 'This will take some time...'
                $(ingpth)report -f/tmp/$rpt.out$$ candeval $rpt
                lpr -q "${parprnt}" /tmp/$rpt.out$$
                echo 'Report printing on draft (parallel) printer' ;;

#       f, F, or word starting with f or F are valid
#       runs report, puts output in temporary directory, print
#       from temporary directory to serial printer
#       $$ - associates the report with the logged in process
        [fF]* ) echo 'This will take some time...'
                $(ingpth)report -f/tmp/$rpt.out$$ candeval $rpt
                lpr -q "${serprnt}" /tmp/$rpt.out$$
                echo 'Report printing on final (serial) printer' ;;

#       user entered invalid choice
        *) echo $prchoice ' invalid entry' ;;
    esac
fi

# reset ok flag so can print more reports
ok=yes

done
)
```

ksabank () {

```
# shell script name. ksabank
# R. Loffman
# October 1986
```

```
# this is the menu for the user to enter the KSA ITEM BANK
```

```
# calls:
# ksamaint
# ksalink
# ksareports
# ksaitemanalysis
# candmenu

# Variable pth is exported from mainmenu It is the path which describes
# where all shell scripts are located. Set in mainmenu.

# Variable clear is used to clear the screen It is exported from
# mainmenu

while true
do
    clear

    # display menu

echo '
*****
(KSA)
KSA ITEM BANK
MAIN MENU
0 - Return to CANDIDATE EVALUATION SYSTEM MAIN MENU (CE)
1 - Maintenance (KSA or TASK DESCRIPTION) (KSA010)
2 - Linkage of Series and Grade to KSAs and TASKs (KSA020)
3 - Reports (terminal and printers) (KSARPT)
4 - Test Item Analysis Screens (KSA030)
*****

Please enter selection and press RETURN \c'

# read and process selection

read choice
echo
case $choice in
# user wants to return to CANDIDATE EVALUATION SYSTEM MM
ce : CE : 0) candmenu ;;

# user wants to add, edit, delete, or retrieve a KSA or TASK
ksa010 : KSA010 : 1) ksamaint ;;

# user wants to construct a Series-Grade-KSA-TASK item
ksa020 : KSA020 : 2) ksalink ;;

# user wants to print reports involving KSAs, TASKs or Linked Items
```

Feb 5 12 27 1987 candidate Page 11

```
ksarpt : KSARPT : 3) ksareports ..
#      user wants to review item analysis
ksa030 : KSA030 : 4) ksaitemanalysis ..
#      user entered invalid choice
* : echo $choice ' invalid entry' ..
esac

done
)
```

cardmenu () {

```
# shell script name candmenu
# R Loffman
# September 1986
```

```
# this is the menu for the user at the unix level
# to enter candidate evaluation system
```

```
# calls:
#      ksabank
#      ksamaint
#      ksalink
#      ksastemanalysis
#      ksareports
#      mainmenu
```

```
# Variable pth is exported from mainmenu It is the path which
# describes where all shell scripts are located. Set in mainmenu

# Variable clear is used to clear the screen It is exported from
# mainmenu
```

```
while true
do
```

clear

```
# display menu
```

```

echo '
*****
(C)
CANDIDATE EVALUATION SYSTEM
MAIN MENU

0 - Return to PERSONNEL OFFICE MAIN MENU (ACP)

1 - KSA ITEM BANK MAIN MENU (KSA)
2 - CREDITING PLAN BANK MAIN MENU

```

*

```
        Please enter selection and press RETURN: \c'

#       read and process selection

        read choice
        echo
        case $choice in
#         user wants to return to PERSONNEL OFFICE MAIN MENU
            acp : ACP : 0) mainmenu ;;
#         user wants to enter KSA ITEM BANK
            ksa : KSA : 1) ksabank ;;
#         user wants to enter CREDITING PLAN BANK
            2) echo 'not implemented yet' ;;
#         user wants to go directly to KSA ITEM BANK MAINTENANCE MENU
            ksa010 : KSA010 ) ksamaint ;;
#         user wants to go directly to KSA ITEM BANK LINKAGE MENU
            ksa020 : KSA020 ) ksalink ;;
#         user wants to go directly to KSA ITEM BANK ANALYSIS MENU
            ksa030 : KSA030 ) ksaitemanalysis ;;
#         user wants to go directly to KSA ITEM BANK REPORTS MENU
            ksarpt : KSARPT ) ksareports ;;
#         user entered invalid choice
            *) echo $choice ' invalid entry' ;;
        esac

done
)
```

```
mainmenu () {

# shell script name: mainmenu
# R. Loffman
# August 1986

# called by login shell /bin/candidate

# other files called:
#   candmenu
#   helpp

# variable ingpth is the path to Ingres commands
```


Feb 5 12:27 1987 candidate Page 14

```
#           user entered invalid choice
#) echo $choice ' invalid entry' ;;
        esac
done
}
```

```
# this is what the user logs into
# it calls mainmenu which directs all activity

mainmenu
```

VITA

A native of Williamsville, New York, Regis S. Loffman graduated cum laude from the State University of New York at Buffalo with a Bachelor of Arts degree in Mathematics in June 1977.

Upon completing her undergraduate degree, she taught Mathematics at the high school level in North Carolina for three years. In 1980, after moving to Tennessee, she began part-time studies in Computer Science at East Tennessee State University.

She was employed as an applications programmer by the Knox County Automated Systems Division from 1981-83. In January 1983 she entered The Graduate School of The University of Tennessee, Knoxville. She received the Master of Science degree with a major in Computer Science in June 1987.

The author was awarded the Certificate in Data Processing (CDP) by the Institute for Certification of Computer Professionals in May 1986. She is a member of the Institute of Electrical and Electronics Engineers (IEEE) and is cofounder of the Oak Ridge - East Tennessee Chapter of the IEEE Computer Society.

Since January 1984, the author has been employed by Martin Marietta Energy Systems at the Oak Ridge National Laboratory* as a computing analyst.

*Operated by Martin Marietta Energy Systems, Inc. for the U. S. Department of Energy under contract no. DE-AC05-84OR21400.