1

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Services des thèses canadiennes

Ottawa, Canada
K1A 0N2

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not film.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

**THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED**

**LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE**

National Library
of Canada

Bibliothèque nationale
du Canada

CANADIAN THESES
ON MICROFICHE

THÈSES CANADIENNES
SUR MICROFICHE

0-315-22186-0

NAME OF AUTHOR/NOM DE L'AUTEUR _____

TITLE OF THESIS/TITRE DE LA THÈSE _____

UNIVERSITY/UNIVERSITÉ _____

DEGREE FOR WHICH THESIS WAS PRESENTED/
GRADE POUR LEQUEL CETTE THÈSE FUT PRÉSENTÉE _____

YEAR THIS DEGREE CONFERRED/ANNÉE D'OBTENTION DE CE GRADE _____

NAME OF SUPERVISOR/NOM DU DIRECTEUR DE THÈSE _____

Permission is hereby granted to the NATIONAL LIBRARY OF
CANADA to microfilm this thesis and to lend or sell copies
of the film.

The author reserves other publication rights, and neither the
thesis nor extensive extracts from it may be printed or other-
wise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈ-
QUE NATIONALE DU CANADA de microfilmer cette thèse et
de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la
thèse ni de longs extraits de celle-ci ne doivent être imprimés
ou autrement reproduits sans l'autorisation écrite de l'auteur.

DATED/DATE _____

SIGNED/SIGNÉ _____

PERMANENT ADDRESS/RÉSIDENCE FIXE _____

21 TURBET CRT
KANATA CRT
K2A 2A1

A User Agent For Multiple Computer-Based
Message Systems

by

© Russell McDowell, B.Eng.

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfilment of the requirements for
the degree of
Master of Engineering

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
Ottawa, Ontario

March 1984.

The undersigned recommend to the
Faculty of Graduate Studies and Research
acceptance of the thesis

"A USER AGENT FOR MULTIPLE COMPUTER-BASED
MESSAGE SYSTEMS"

submitted by
Russell McDowell, B.Eng. in partial fulfilment
of the requirements for the degree of
Master of Engineering.

Thesis Supervisor

Chairman,
Department of Systems and
Computer Engineering

Carleton University
Department of Systems and Computer Engineering
April 1984.

Abstract

CCITT draft standards for electronic messaging systems now exist. These are based on the User Agent / Message Transfer Agent concept. However the existing message systems are not designed on such a model and for the next few years it will be necessary for an office worker to be able to utilize both types of systems.

This thesis reviews the design of a User Agent as an intermediate step between the existing heterogeneous message systems and the future when all message systems will present the same interface. The thesis outlines the difficulties encountered in an experimental implementation and the solutions adopted. Many of these would also be applicable in providing mediated access to services other than messaging. The User Agent designed in this thesis resides in a personal workstation and utilizes existing message systems as message transfer agents.

Acknowledgments:

I wish to express my gratitude to Professor A. R. Kaye, who suggested the initial topic for this thesis. He was most generous with his time, and patient in guiding my work around many obstacles, and finally to conclusion.

I also owe a great deal to my wife for her understanding and emotional support as well as for her help in reading and editing the numerous preliminary drafts.

Finally, I wish to thank Bell Canada for providing access to the ENVOY-100 message system.

CONTENTS

1	Introduction	1
1.1	Office Automation	1
1.2	Electronic Mail	3
1.3	Professional Workstations	4
1.4	CBMS Architecture	6
1.5	Aims of This Thesis	8
1.6	Methodology	10
1.7	Structure of Thesis	10
2	Analysis of Existing CBMS	12
2.1	Characteristics of Use	12
2.2	Disadvantages	14
2.3	Common Facilities	15
3	High-level Design of the Experimental UA	19
3.1	User's Conceptual Model of the UA	19
3.2	Functional Specification	21
3.3	UA Architecture	24
3.4	Typical User Scenario	27
3.4.1	Starting the UA	28
3.4.2	Getting external messages	28
3.4.3	Local reading	29
3.4.4	New messages	30

3.4.5: Sending messages	30
3.4.6: Filing of messages	31
3.4.7: Directory command	31
4 Implementation Plan	34
4.1 File Structure	34
4.1.1 CBMS Command File	34
4.1.2 Message Files	36
4.1.3 Registered CBMS File	36
4.1.4 Correspondents Data File	37
4.2 The Communication Process	37
4.3 Portability and Hardware Dependencies	39
5 Implementation Problems	42
5.1 File I/O	42
5.2 Inter-Computer Communications	43
5.3 Flow Control	46
5.4 Use of Other Tools	48
5.5 Analysing Response	52
5.6 Non-printing Characters	53
5.7 Sending Copies	53
5.8 Hardware Dependencies	54
6 The Experimental UA: A Summary	57
6.1 Further Work on the Experimental UA	57
6.2 Extensions of the Concepts Developed	60
6.3 Conclusions	63
References	66
Appendix A: Portability Issues	68
Appendix B: User Manual	71

Appendix C: Data File Formats	77
Appendix D: System Structure Charts	97
Appendix E: System Code Files	106
Appendix F: Sample Run	148

Chapter 1

Introduction

This thesis concerns itself with one aspect of the problems of developing a personalized "tool box" for the office of the future. In particular, this thesis outlines the analysis and design of a user agent which provides a generalized electronic mail service, operates in a microcomputer, and utilizes external electronic mail systems as message transfer agents. In doing so, it investigates and finds solutions for many of the problems that would be encountered in designing facilities for computer-mediated access to many other types of public or private services available through data networks. Databases of all types fall into this category of services. An additional objective is to explore methods of providing such a specialized tool while achieving integration with tools already available in the microcomputer environment such as the editor.

1.1 Office Automation:

If one is to deal with the topic of office automation, it is appropriate to clarify what one means by "office automation". The most significant use of computers is to enhance human capabilities, not to actually replace office personnel. Office automation is not a solution, it is a medium [1]. Defined as a medium, it is both a means of

effecting or conveying something and a condition in which something may function or flourish. Another definition of office automation is the following:

"Office automation is the use of information processing technology in an office environment to create, process, store, retrieve, use, and communicate information to improve performance of managerial, professional, technical, administrative, and clerical tasks." [13]

In other words, the basic aim of automating the office is to increase the productivity of the office worker. To date the most widespread use of office automation has been to facilitate secretarial and clerical functions with such systems as word processors and point of sale terminals.

However, the great promise of the office of the future concept lies in its application to the office managers and professionals. It is stated elsewhere [3] that this category accounts for 78% of office costs compared to 22% for the secretarial / clerical staff. Therefore any increase in productivity by this class of "knowledge worker" would greatly outweigh a corresponding increase by clerical staff in terms of overall costs. The reason that office automation has almost always concentrated on the secretarial and clerical staff is that the types of jobs that these people are involved in are well defined and very repetitive. It is therefore easy to develop tools or systems that assist them in their jobs. The main stumbling block until now in the

assistance of the knowledge worker has been in the greater variety of functions that they perform.

With the advent of the microcomputer, this is changing. It is now possible to put at a manager's disposal a personal computer capable of being adapted to many different tasks, simply through the use of different software. However, it should be noted that the goal in using computers is not to replace the office worker, but rather to assist him. Without great improvements in the area of artificial intelligence, it will not be possible to replace a reasoning person with a machine. Therefore the machine will simply become an extension of the person to be used to increase his capabilities. As an example, consider the widespread use of calculators. It is possible to do arithmetic without them but their use increases output, and confidence in the results. At the same time, without a knowledge of how to interpret the results a calculator is worthless.

1.2 Electronic Mail:

If the purpose of office automation is to aid the office worker, then first one must determine the areas of broadest application. The three primary objectives of an electronic office for managers are: [9]

1. spend less time on paperwork.
2. decision - making capability be improved.
3. human communications be improved.

The literature [e.g. 16] on the subject reveals that a typical manager may spend between 50-90% of his time communicating with his fellow workers. Given this figure, better communications would obviously help managers to derive the maximum benefit from office automation. Thus any system that is designed for the office of the future will require an electronic mail facility.

There is a wide diversity of opinion on what comprises electronic mail. Some definitions encompass any electronic means to deliver a message and thus include such services as facsimile, voice messaging, or even the telegraph and telephone. This thesis concerns itself with a much narrower view of electronic mail. In particular it concentrates on computer based message systems (CBMS) which have a store-and-forward capability for alpha-numeric message text.

1.3 Professional Workstations:

The cost of computer hardware has plummeted in recent years. The widespread introduction of the microcomputer has made it feasible to dedicate a personal computer to every office manager and professional worker. The field is still

changing, and while it may not yet be possible to determine the exact configuration of such systems, certain generalizations can be made. The heart of these microcomputers will be at least a 16 bit processor, or perhaps a 32 bit as they become available. The system will have local mass storage capability. This may be in the form of floppy disks, but more likely would contain a winchester disk and perhaps bubble memory. These systems are meant for groups of knowledge workers, therefore they must have the capability of communicating with other workstations and mainframe computers. As a result, the workstation configuration will have either a telephone line connection, or more likely, either a private business exchange or local area network facility. In the latter case, the probable use of gateways would allow a user access to the public data networks such as DATAPAC and TELENET. From these public networks most mainframe computers could be accessed.

As mentioned, the advantage of having the personal computer is the capability of adapting it to different tasks. Therefore it is reasonable to predict that office workers will accumulate a collection of software tools to aid them. However, many of these will incorporate or use similar subfunctions such as database management or editing. Typically each tool now "re-invents the wheel" and duplicates these subfunctions. This results not only in needless duplication of effort but in an inconsistent

1

interface to the user and tools that do not take full advantage of the system facilities. What is required is a strategy whereby new tools make use of those already developed.

1.4 CBMS Architecture:

Most of the existing message systems provide a facility which permits a user to prepare a message and to have it sent to recipients on the same CBMS. In general these systems are self-contained. They provide all the editing and filing facilities that are necessary for creating, handling and storing messages. This places restrictions on the interface that is provided to the users. Most CBMS assume that the user has a line oriented terminal and do not provide higher facilities such as full screen capabilities, which would be utilized by a smaller audience. They also assume that the CBMS is dealing directly with a human user; they are not equipped with protocols for machine to machine transactions. They ignore the fact that, today, the user may well be utilizing a powerful professional computer incorporating all the editing and filing facilities that are needed.

The proposed CCITT standard on message handling has split the two main functions of messaging apart. The actual interaction with the user is handled by what is termed a

user agent (UA). The originator of a message interacts with this UA in order to create, edit and review his message. When the user wishes to send his message the UA submits the message to the message transfer system (MTS) on behalf of the user. The MTS is comprised of one or more message transfer agents (MTA) which may be located on multiple computers. These MTA co-operate in order to pass the message from the originator's to the recipient's UA. Once the message is received, the UA will interact with the recipient in order that he may read and respond to the message. Thus there is a very clear distinction between the process of creating and reviewing messages and the actual transmission of them.

The CCITT work also makes a clear distinction between the envelope of the message and the contents of the message. The MTS looks only at the envelope of the message and is not concerned with the format of the contents. Thus it is possible to send messages, which may contain binary or graphical data, using the same CBMS as text messages. The content of the message is likewise divided into two parts: a header and the actual message. The UA can tell what resources (eg: graphical output) are required to deliver a message from the header information and can take appropriate action.

1.5: Aims of This Thesis:

This thesis is an initial attempt at designing and building a UA capable of mediating access to multiple, external CBMS which do not conform to the CCITT model. Such a UA could exist in either a personal workstation or as a special server on a local area network. Such a UA could also be used as the basis of a gateway facility on a standard MTA to enable it to communicate with non-standard CBMS. The problems investigated to achieve mediated access to CBMS have much in common with those that would be encountered in mediated access to other types of services.

The UA will use the local tools for most of its functionality and will only access the external CBMS for the required facilities. These are typically the actual reading and sending of the messages (ie: using the CBMS as a MTA only). This method of operation would have several benefits to a user. The main one is that he would be able to use the facilities that he is most familiar with and which are best suited to his environment. For example, he would be able to use a full screen editor for the creation of messages and his own file system for their storage. Second, he would be able to reduce the costs of and dependence on mainframe computers which would only be used for sending and receiving messages and not for editing and filing them. Third, the UA

would allow for better, more consistent user interface and help facilities than on many external CBMS.

There are three basic areas that are of concern in this type of application. The first is the inter-computer communication problems and how to talk to other computers without high level protocols. The second area of difficulty is how to reconfigure UA software for the various external CBMS without the need for program modifications and rebuilding. The third basic area involves the aforementioned use of existing tools. While these issues have been brought up in the context of electronic mail, they are also common to the other areas such as databases.

The above objectives lead to two main results. The first of these is an analysis of problems that have been raised and potential solutions to them. The second result is an experimental version of a UA. This UA was designed to test some of the solutions proposed and as a mechanism for uncovering any problems that might not have been considered. Needless to say, there are various limitations and design considerations that are dictated by the type of hardware, operating system and language that were used for the experimental UA. These will be dealt with more fully in the chapter on UA design.

1.6 Methodology:

The first step was to review several different CBMS now in existence. From this information a subset of necessary common functions were derived. From this list, and an analysis of what a knowledge worker may expect, a set of user commands and capabilities to be provided by the local machine was drafted. These were then implemented in the experimental UA in order to check the overall design and the usability of a system developed with these functions.

1.7 Structure of Thesis:

Chapter two deals with an analysis of the major external CBMS that have been studied and the functionality that should be provided in an electronic mail system.

In the third chapter the functional specification and the high level system design of the UA is presented. In addition the chapter contains a typical scenario of use and how the UA performs its tasks.

The fourth chapter deals with the specific design details and how the high level design has been implemented. It lists the primary functions that have been implemented and what they do in terms of UA functionality. That section includes a discussion of various software engineering

details, reasons for choice of language and machine for implementation, operating system choice and limitations imposed by it. In addition it covers the main functions within the UA.

The fifth chapter delineates the implementation issues that have been uncovered during the design of the UA. It outlines the problems encountered, alternative solutions to them and solutions that were adopted for purposes of this thesis.

The last chapter reviews the objectives of the thesis and how the UA has tried to attain them. It looks at implications of better solutions to the problems that were encountered in terms of computers systems architecture (eg: operating systems that allow program overlays, high level protocols between programs, etc.). The last part of the chapter details further work that may be done on this particular implementation to evolve it into a better tool for the office of the future environment. The appendices to the thesis contain structure charts of the UA, sample data files and how they are set up, program listings and a discussion of portability issues in moving the implementation to another computer system.

Chapter 2

Analysis of Existing CBMS

The analysis reported in this chapter was done in order to compile a list of functions that should be provided by a CBMS in general. Three different CBMS were used in the experiment. The first was ENVOY-100, a public system offered by the Trans Canada Telephone System. ENVOY was only accessible through the DATAPAC network. The second system was COCOS, a private, corporate message system running on an IBM mainframe over local, dial-up ports. The third was the mail system offered by The Source, a public information utility oriented primarily to the hobbyist market and accessed via DATAPAC and TELENET.

2.1 Characteristics of Use:

CBMS combine the best features of paper mail systems while also offering some of the advantages of electronic means of communication such as the telegraph and telephone. One advantage is that an electronic message enables one to obtain a written record of what has transpired. Another advantage of CBMS is that it is possible to send copies of the message to other people and therefore is more efficient than using a telephone to disseminate information. CBMS offer the option of which messages to deal with first, whereas the telephone does not.

A difficulty with using the telephone for communication is the inherent real-time aspect of its usage. The problem of "telephone tag" where the people involved never seem to be able to find one and other in the office is well known. This problem is aggravated when trying to contact people in different time zones. CBMS do not require the response of both parties at the same time and therefore eliminate these problems.

Another problem of telephone communications is that it normally interrupts other work that the recipient of the call is busy with. He must set aside whatever he was working on to deal with the phone call and then try to resume his line of thinking on the original task. This can lead to a loss of productivity greater than the time occupied by handling the actual call. With CBMS the worker can decide the most convenient time for him to handle the messages and can better deal with both the actual messages and his other tasks. In addition, it is possible for a person to check his messages from any terminal. He does not have to be at his office. This feature facilitates decentralization while maintaining close communications. Another advantage of CBMS is the virtually instantaneous speed of delivery. Once the message is posted by the sender, the recipients are able to read it regardless of geographic dispersion. This is one of

the main advantages that such systems have over the regular mail.

2.2 Disadvantages:

As with any system meant for general communications, it will be effective only in direct proportion to the number of people that may be reached by it; the critical mass effect[11]. There now exists a multitude of CBMS and if more are not to develop as "electronic islands"[2] then the means must be found to interconnect them. A disadvantage with the use of electronic messaging is the different user interface. Not only does the CBMS editing function differ from the interface provided to the user from his local editor, but in the case of using multiple CBMS they will typically all present a different set of commands to handle messages for him.

Another problem with using CBMS to date has been that a user wishing to send a message to users who happen to be on two different CBMS must create the same message on both of them. This leads to a fragmentation in the filing system of the originator of the message as he must now check both CBMS for the originals and any follow up messages dealing with the topic.

In addition, computer systems generally have a reputation of not presenting a friendly user interface nor of offering well designed help facilities. If a user has a personal workstation then it is more cost effective for him to use his local workstation for whatever he can and use the CBMS only for its main function, that of providing a centralized location to deposit messages for other workers.

2.3 Common Facilities:

While the various CBMS that exist all operate in different ways, there is as may be expected, a fundamental set of functions that exist on all of them. The basic requirements of a CBMS is that it must be capable of creating, sending, and receiving messages at a user's convenience. While these requirements constitute the minimum, most commercial CBMS offer far more.

In general the CBMS try to emulate a typical desk setup and model their functions after familiar office procedures[5,8,12]. Any incoming messages are deposited into a user's "inbox". It is from this default file that messages are retrieved, read and acted upon. The CBMS generally use the idea of a "workspace" as the equivalent of a desk top. It is this area that is used to create and modify messages. Some of the CBMS also have the concept of an "outbox". Once a message is ready to be sent, it is removed from the

workspace and deposited into the outbox. On other CBMS the message is sent as soon as it is removed from the workspace. The advantage of the outbox concept is flexibility; after a message has been acted upon, a user can stop it from being sent. The main drawback of this approach is that the user must take explicit action to send it from his outbox and there may be no warning mechanisms to remind a user of unsent messages when he logs off. Some CBMS avoid this problem by automatically transmitting immediately on receiving a send instruction but permitting later retrieval and cancellation unless, of course, the message has already been read by the recipient.

Another similarity in the CBMS to the physical environment of the office is the provision of a filing cabinet. Most CBMS will allow a user to save a message in a file for later reference. Users may save copies of related messages in separate files for ease of future reference. Another common facility that is provided in CBMS is the capability to use distribution lists. These are files of recipient names that are to be used when circulating a message.

A facility that is quite common in commercial CBMS is the concept of bulletin boards. These are public areas where any user may post and read notices. In the case of large

CBMS these boards are normally divided into different areas based on subject matter or organizational structure.

Another capability of CBMS is the option of sending registered messages , or notifying the sender of delivery. This form of sending a message allows an originator to be assured that his message has been read. In order to receive such notification, the sender must specify the option when the message is originally sent.

Another capability that closely parallels the above concept allows the sender to check if a recipient has any messages that have not been read. This differs from the registered message in that an originator may check some time after the message has been sent, if he has not received a reply. Although he is able to see that unread messages are in the recipient's inbox, he cannot know whether his is among them.

Most CBMS also allow a user to be selective on his mailbox scans. For example, there usually exists the capability to limit a scan command by subject, date of message, or originator of the message.

Other CBMS allow originators of message to address them to recipients which do not have access to the CBMS. For example, an originator can use COCOS to send a message to

anyone within the company and it will be delivered through the internal mail system rather than being deposited in his computerized inbox. Similarly, it is also possible to send messages from ENVOY-100 and have the Post Office deliver them through the regular mail delivery.

Another facility that is provided within private CBMS is the option of having a copy of specific messages printed. On the COCOS CBMS these messages are printed in the nearest mailroom and delivered through internal mail. In addition, it is also possible to use the COCOS system to check the names of employees within the company.

Chapter 3

High-level Design of the Experimental UA

3.1 User's Conceptual Model of the UA:

The conceptual model adopted for the UA is depicted in figure 1. When the user logs onto the UA his new mail is located in his inbox. This is the default area from which messages are retrieved to be acted upon. The UA provides a mechanism whereby the user may reset this default box to any one that has been previously created with saved messages. When the user wishes to read and possibly take action on a message it is transferred to the workspace area. This corresponds roughly to a user's desk top in a typical office environment. Outgoing messages to be sent via CBMS are stored in the outbox until the end of the user's session, when they are forwarded. The UA also provides the utility of a "filing cabinet" where the user may save copies of any messages which have been received or created. This model is common to many of the existing CBMS. However the user of the UA will also be aware of the fact that the local workstation must collect messages and send them via external CBMS. In addition the user will normally be aware of the local tools (in this case the editor) that he is invoking from the UA.

FIGURE 1

USER'S CONCEPTUAL VIEW

3.2 Functional Specification:

Based on the analysis of the various CBMS and the system requirements it was concluded that the following functions would be required in the experimental user agent.

3.2.1 Specify external CBMS hosts

A user must be able to identify external CBMS to the UA. This must include information about logging on and off the CBMS, issuing commands to them as well as how to analyse and interpret the responses from them. The mechanisms by which this is done are described in detail in a later section.

3.2.2 Maintain directory of known correspondents:

On each external CBMS the list of potential recipients of messages is usually the same as its total range of users. Messages may only be sent to users registered on that CBMS and one addresses messages to a user by the same identifier that he uses when he logs on. The UA on the other hand must have information on which external CBMS to use to reach a given correspondent. It therefore maintains a file of known correspondents. One of the benefits of this function is that the UA can translate external user identification names

(userid's) at the same time and thus allows the office worker to use meaningful names in his correspondence.

3.2.3 Get messages:

One of the main requirements of the UA is to retrieve messages from external CBMS and transfer it to a local inbox. This function allows either a mode of retrieval by cycling through a list of registered CBMS or from a single one, at the user's discretion.

3.2.4 Review messages:

The user must be able to list the contents of the default inbox. In addition he must be able to change the default inbox to any of the others which contain saved messages. After doing this the user should then be able to read messages from those boxes selected in any order.

3.2.5 Take action on messages:

1. Reply to a received message. The UA should allow automatic creation of the reply header, including address and subject.

2. Forward a received message to a new recipient.

The UA must allow the user to annotate the

original message.

3. Create a new message.

4. Identify multiple recipients to receive copies of messages.

3.2.6 Send messages:

When the user terminates a session by exiting the UA, it should automatically send the contents of the outbox. This is done by logging onto external CBMS, transferring appropriate contents of the outbox and logging off the CBMS until all messages have been sent. It is also necessary to allow a user to exit the UA without sending any messages if he so wishes. In addition, the UA allows the user to send the messages from the outbox by explicit command without exiting.

3.2.7 File messages:

Store copies of incoming or outgoing messages in the user agent's file system.

3.2.8 Delete messages:

Delete messages from either the default in and out boxes or from user-defined boxes.

3.2.9 Register an external CBMS:

Register an external CBMS for automatic collection of messages by the UA.

3.3 UA Architecture:

As mentioned in the introduction, the main function of the personal workstation as a user agent is to mediate the access to external CBMS via local intelligence. This will allow the office user to maximize his benefit from local facilities while also maintaining a consistent interface across various external CBMS. Such an approach implies three capabilities:

- A. The ability of the two computers to converse without the use of established protocols.
- B. The ability to adapt the UA to deal with new CBMS without the need to reprogram it.

- C. The ability to allow the user access to local system tools from the environment of a program.

Figure 2 depicts how the various system components work together to achieve these goals. As shown, the user only interacts with the UA. When an external CBMS has to be accessed it is done automatically by the user agent. In order to allow the UA to gain access to the various CBMS it is necessary for it to have knowledge about the command syntax for them. This information is contained in the "CBMS command" file. In addition to the syntax of the commands, the user agent requires information that is peculiar to each office worker such as his userid and login passwords for each of the CBMS. This information is typically of a more sensitive nature than the command syntax and would therefore be stored in an account data file which would normally be protected by encryption in a fully developed UA. Thus within an organization each user would have the same copy of the CBMS command file, but the account data file would be modified for each user. However, in the experimental UA the only data in this file is the password that is used to log onto the UA itself. The rest of the account data is incorporated into the command sequences in the commands file described later.

Other files that are used by the UA are the actual message files. When the UA has logged onto a CBMS it will

FIGURE 2
OVERALL SYSTEM INTERACTION

access the outbox file to retrieve the envelope and message data in order to send a message. Likewise it will deposit external messages into the inbox file. In addition, whenever a user modifies or creates a message the UA will deposit it into the outbox.

Since it is part of the UA design to use the local system tools whenever common functions, such as editing, are required, the diagram also shows that the UA will access the editor whenever the user is editing messages. This is done through facilities provided by the operating system. Currently the only system tool that is used by the UA is the system editor although there are no fundamental restrictions on using other facilities as well.

Finally there is a "help" file. This file contains information on the error messages and on-line help facilities that are provided to the user. Whenever a system error is discovered or the user requests help, the UA will use a code number to find the appropriate message to output to the user.

3.4 Typical User Scenario:

The following are brief descriptions of how the UA implements the basic facilities that were outlined using the

data files mentioned above. A full list of the available commands is given in Appendix B.

3.4.1: Starting the UA:

On starting up the UA (either by explicit command, or possibly a built in "boot" command) the user is requested for a password. On receiving this input the UA verifies it against a copy in the account data file. The UA allows three incorrect passwords and then terminates if the proper password has not been input.

3.4.2: Getting external messages:

After the UA has validated the user, the next step is to (automatically) logon to the various registered CBMS and retrieve the messages. The algorithm for doing this is for the UA to open the registered CBMS file and read the list of external CBMS. Once this is obtained it cycles through each CBMS, using the CBMS commands file to give it the appropriate input sequence to dial up and log onto a given CBMS. Normally the login commands will contain the code requesting parameter substitution. Encountering this, the UA reads the appropriate user name and password from the personal data file and substitutes it at appropriate spots. Logging in, the UA uses the CBMS command file sequences to check for messages. Finding a message, the UA reads it from

the external CBMS, stores it locally into the inbox file and loops back checking for more messages on this CBMS. Once all messages on a given CBMS have been read and locally stored, the UA uses the commands file to determine how to get out of the external CBMS and log off the mainframe computer. The UA loops back over this procedure until all messages have been received from all the registered CBMS.

3.4.3: Local reading:

The next phase in UA usage is for the user to check his inbox and reply to the various messages. The first command that is used is the scan command. This command opens the inbox file and prints out the envelope, skipping over the actual text of the messages. The envelope consists of the prime recipient's name, originator's name, names of recipients for copies, and the subject of the message. The next thing the user wishes to do is to read a message. At this point the UA re-opens the inbox, skips all the messages not required, and then displays the envelope and text of the requested message. While this message is being displayed the UA also stores it in a temporary file. After the entire message has been read the user is asked for an action to be taken. If the user ignores the query, the temporary file is deleted. If he chooses to draft a reply, then the UA creates another temporary file. The UA then creates the subject line and envelope information for the reply. If the user answers

that he wishes to forward the message then the UA generates an envelope and copies the original message text from the first temporary file into the second one. In both of the last two cases the user then goes into edit mode (the system editor) where he drafts his reply, or annotates the original message. On exiting the editor, the UA automatically inserts the temporary message file into the outbox so that when the user logs off the UA the message will be sent.

3.4.4: New messages:

The creation of new messages is done in an analagous fashion to the reply command. The UA first puts the user into the edit mode to allow him to compose his message. The UA then prompts the user for the envelope data after he has terminated the edit. It then inserts the new message and envelope into the outbox to be transmitted on the next send directive.

3.4.5: Sending messages:

Sending of the messages can be done either automatically when the user exits the UA or by explicit command. In either case the function is implemented in the following manner. The UA opens the outbox file and reads the envelope data. From this data it is able to determine the name of the recipient. The UA goes to the file containing

the list of known correspondents to determine which CBMS the message should be sent on. Once this is determined the UA then uses the CBMS commands file in order to logon to the appropriate CBMS and send the message. The UA loops at this point, sending messages to all recipients listed for copies. The UA then loops back to the next message in the outbox and repeats the above procedure until all messages in the outbox have been sent.

3.4.6: Filing of messages:

The UA also allows a user to store messages in a different box than the default inbox or outbox. When a user reads a message he is given an option of filing the message in a user specified box. The UA checks to see if it already exists. If it does then the new message is appended to the already existing box. If it does not exist then a new one is created and the message inserted into it. These message files have the same format as the default files and therefore it is possible to redefine the inbox to one of these and use the facilities of the UA to scan, read, and take action on any messages that may have been saved in them.

3.4.7: Directory command:

The user may obtain a directory of all known correspondents. This gives a list of all the recipients that the UA is capable of sending messages to. The first line of an entry is the name that is used locally by the UA. The second is the external CBMS that the UA uses to send messages to him. The third line is the name that the external CBMS requires to send messages to the recipient. The rest of the entry is optional and may include such items as job titles, phone number, mailing address, etc.

3.5 Help Facility:

In addition to the commands associated with electronic messaging the UA also provides a help facility. When building tools for the knowledge worker it is important to remember how he differs from clerical users. The knowledge worker will tend to be a discretionary, casual, non-computer oriented user. Unlike clerical staff he will probably not be using the workstation all the time, day after day, nor will he be able to spend extended periods in training to become familiar with the application. Therefore it is important that the UA provide an easy means by which the user may explore the UA facilities.

The first feature of the help facility provided on the UA is the provision of context dependent help. Whenever the UA is requesting input from the user he may respond with a

'?'. The UA will then locate the appropriate message in the help file and display it.

The second feature of the help facility is that all error and help messages are contained in one text file. These may be modified without the need to change the program. Therefore it is easy to change the help given to the user if it is found that some of the messages aren't clear or misleading. In addition, it has the side benefit that the overall code size for the application is smaller in that all the text for the messages does not have to be imbedded in the code.

Thirdly the user is able to scan up the chain of help messages. This allows him to trace back if he gets confused while responding to a command. With some enhancements this facility can be expanded to give different messages appropriate to user sophistication level. Also it could be enhanced to allow a user to traverse down the help messages, allowing a user to explore the facilities of the UA without actually executing commands.

Chapter 4

Implementation Plan

An experimental version of the UA defined in chapter 3 was implemented on an 8080, CP/M based system. The following paragraphs outline the techniques that were used. While it is accepted that the choice of CP/M for the trial was not ideal, it was chosen simply for availability.

4.1 File Structure:

This section describes the data files that are necessary for the operation of the UA and their internal structure.

4.1.1 CBMS Command File:

This file contains the information necessary for the UA to be able to access and communicate with the various external CBMS without the use of a high level protocol.

It is roughly based on the idea of macro instructions. From the analysis of the existing CBMS and the functional specification of the UA it was possible to draw up a list of macro instructions that are required when communicating with the external CBMS. There is not necessarily a one-to-one correspondence between a macro required by the UA and

specific commands on the CBMS (for example, the login sequence). The following macro instructions are included in this file:

A) Logon

- number to dial
- possible datapac commands
- host login sequence
- access to CBMS

B) Logout

C) Check mail

- commands to tell if any (more) mail on CBMS.

D) Read mail

- read a message from the CBMS.

E) Send

- automatically enter a message on the external CBMS in order to send it.

F) End send

- get out of enter mode of the external CBMS and send the message just created.

G) Analyse

- information on how to analyse the header of a message read from the CBMS.

H) Check end

- information on the response expected from the CBMS when "check mail" is used to check for (additional) mail.

Thus whenever the user wishes to be able to access a new CBMS via the UA he simply adds these commands to a new section in the CBMS commands file. This is currently done using the editor but a more fully developed UA would include a utility which would obtain the information interactively from the user with appropriate prompts. The setting up of the data files is covered more extensively in Appendix C.

4.1.2 Message Files:

All message files used by the UA are of the same format. The inbox and outbox are special cases only in that their default names are known by the UA. Each message file consists of variable length records which make up each message. These records are terminated with an end-of-record delimiter ('0', 7E hex). Within each record the message contains fields for recipient, originator, copies, subject and actual text.

4.1.3 Registered CBMS File:

This file is used by the UA to determine which CBMS to automatically check for messages. Each line of the file corresponds to a section name in the CBMS commands file. Thus the UA knows which section of the commands file to use for the external communications.

4.1.4 Correspondents Data File:

This file contains the information that the UA requires about correspondents in order to know which external CBMS to access to send messages. The file itself contains records (again delimited by '°') on each correspondent that the UA knows about. Within each record there are four fields. The first three, corresponding to the first three text lines, are the correspondent's internal name within the UA, the CBMS through which he can be reached, and his external name on that CBMS. This information is used on sending the outbox in order to tell which CBMS to login to and on receiving messages in order to translate the external correspondents names to the internal ones.

4.2 The Communication Process:

One of the most unique features of the experimental UA is the intercomputer communication. As previously mentioned this has two aspects to it. It must be able to communicate with the mainframe without high level protocols and it must be reconfigurable to allow addition of new CBMS without the need to reprogram the UA.

With the above goals in mind the experimental UA has attempted to achieve them through the use of data files, the

main one being the CBMS commands file. Implicit in the communications of the local computer with the external systems is that the communication lines are in a stable state. Thus the process whereby the UA replaces a human user in communicating with the CBMS breaks down into two major components: determining when the CBMS is ready for the next input and sending the right command or data. The first of these presents some difficulties which are discussed in section 5.2.

The appropriate command sequences for the required functions are obtained from the CBMS command file. Certain of these commands contain parameters which must be substituted with data related to the current transaction. This data is obtained by the UA from appropriate data files and substituted before the line is sent.

However, the main problem in the inter-computer communications is not in the sending of data but in interpreting the responses of the CBMS. Whenever the UA has a response it must analyse, it collects all the data from the CBMS into a temporary file. No attempt is made to analyse the data as it is being received as such processing could potentially cause the loss of characters being transmitted from the CBMS. Once the entire message has been received then the UA attempts to analyse the data.

There are two functions where interpretation of a CBMS response is necessary. The first one is when the UA must tell when there are messages to be read on the CBMS. In this case the UA simply compares the received response to an expected response when there are no messages to be read. If they are the same then there are no messages and the UA proceeds to log off the external CBMS. If the results are not the same then the UA presumes that there are messages to be read and proceeds to the read function.

The next case where the UA must analyse the response from the CBMS is after a message has been read. In this case the UA translates the received message from the format of the CBMS into the internal format of the local message files. Thus the CBMS commands file contains information on how to determine the form of the envelope for each CBMS. This is described in Appendix C on how to set up the necessary data files for the UA.

4.3 Portability and Hardware Dependencies:

The experimental UA has been written in as portable a manner as possible. However, there are some basic assumptions within the system that had to be made. The first is the use of an autodial modem and the protocol established between it and the UA. The second assumption is that of the particular type and address of the I/O port used for serial

communications, and its ability to handle DTR line to the modem. The other assumption, the existence of two disk drives, was made simply because of the small disks used in the experimental implementation. All of these hardware dependencies are well isolated and can be easily modified for different hardware environments.

While the UA makes use of the system editor, no assumptions are made about it and any one could be used. The UA should be able to run under any CP/M environment with the exceptions listed above in hardware. If the UA were to be ported to a different operating system then the following areas should be considered: how the application gets parameters from the command line, a facility to replace the submit invocation of the editor, and different, hopefully better I/O facilities for all the data files that are used.

It is very important that the UA be easily understood and very modular in nature. Therefore it was decided that a high level language should be used for its implementation. In the design of an interactive system, on microcomputers, the use of high level languages would normally cause concern in terms of response time for the user. However the main use of this UA is to display and input text characters. This fact, coupled with the heavy file I/O of the UA (to be discussed later), means that the time penalties should not be noticed by the user.

It was also desirable that the UA should be easily transported to a 68000 microprocessor which would be running the UNIX operating system as this was to be a future configuration at the university. At the time of system design however the microcomputers that were available were 8080 / Z80 systems running the CP/M operating system. Therefore the UA was developed on an 8080 system using the CP/M operating system and the programming language C.

For reasons of portability the hardware dependencies of the serial I/O have been isolated. A further discussion of the hardware dependencies of the system may be found in Appendix A.

Chapter 5

Implementation Problems and Solutions

This chapter outlines the implementation issues and problems that were encountered in developing the experimental UA. It also details some of the alternative solutions to the obstacles uncovered.

5.1 File I/O:

As mentioned in the section 3.3 this application is heavily dependent on the use of files. For ease of implementation of the experimental UA, the I/O of the UA has been done using line oriented text. This means that the UA may not do random access reads or updates on files. These facilities are available under BDS C only in block I/O mode. Thus it would be necessary to write special I/O routines to enable the UA to be line oriented but allow random access. It is foreseen that a better operating system would allow for such I/O operations in an operational implementation. This approach to the file manipulation routines has meant that there is a lot more disk activity than desired. For example, in order to append any information to a file it is necessary to create an intermediate file, copy all the original data to it, copy the appending information, delete the original and then rename the intermediate file to the original name. A similar technique must be used for deleting

messages from files. In the case of reading the data files, such as the command file, it is necessary, in general, to close the file and then re-open it. This is because the order of the commands is not known beforehand and the operating system does not allow the UA to reset a file pointer to the beginning of the file.

A more elegant solution would be the use of a database system for the necessary data files. This would ensure ease of random access, ease of update, ease of organization, and the use of the commands by various programs. The type of data that must be maintained for external communications would seem to partition quite well into such a structure. For example, the experimental UA has access to three external CBMS. For each CBMS it must have a series of records for each command: eg. Logon, Read, Send, Logoff, etc. Therefore adding another external CBMS to the UA would simply be including another entity in the database with an entry for each of the required commands. Such an approach would also facilitate the use of the data files by other programs which also require the capability to talk to mainframe computers.

5.2 Inter-Computer Communications:

The next major problem with the design of the UA is the method of communication between the workstation and the

mainframe computers. Ideally the UA would be capable of telling when the mainframe is ready to accept input. On some mainframes, such as the IBM ones, this is quite easily done by triggering on the X-ON character that is sent whenever it is finished transmitting. However, this is not a standard feature and therefore could not be used for other mainframes in general.

Another alternative was to encode the various prompts within the CBMS commands data file. Thus by matching the prompt against the input received, the UA would know when the CBMS was ready for its next line of input. While this solution may appear desirable there are several problems with it. The first is the number of different prompts that may be involved. The UA would not only require the main prompt line for the CBMS but also the prompts for logging on and off the CBMS, and for the various subsystems such as editing text or reading messages. Another problem associated with such an approach is that there would be very tight coupling between the two computers. If the CBMS changed its prompt then the UA would have to be modified to reflect the change. Such a problem could occur as either a new CBMS release on the mainframe or just by the upgrading of a user's sophistication setting of a CBMS. Such an approach is also complicated by the variable nature of the messages sent by the mainframes when a user logs on. It is typically here that it will display headlines and operating

system notices that may vary depending on the day and time that the user logs on.

The approach that was taken in the experimental UA was to use a timeout technique. For every command that is sent by the UA to the external CBMS there is a delay time associated with it. Thus when a command is sent, it will wait in a delay loop for the specified amount of time before declaring the command sent. Any response character from the CBMS will reset the delay count. Thus the delay is not for the amount of time to execute the command but rather for the amount of time that one should delay from the last response character before assuming that the command is completed. This technique is an improvement over the one mentioned above in that it does not rely on specific responses from the CBMS. However, it does presume that the commands are entered in the right sequence. It also implies that the response of the UA will be slower in that it now must timeout on every input that it sends rather than the immediate response an interactive user could give. The technique also has the problem that the timeouts must be of a sufficient duration that a heavily loaded CBMS will have had enough time to respond to the input. This is especially a problem on such hobby systems as The SOURCE where the number of users can be high and the corresponding response delays quite long at times.

5.3 Flow Control:

There is also a potential problem with the communications of the two computers in flow control. When the CBMS has a particularly long message it may cause the UA to write to disk. With an ideal operating system, where the developer is able to write a multi-tasking program, it would be possible for the UA to have an interrupt service routine which would be able to buffer the I/O activity with the CBMS. Unfortunately, the CP/M system that the experimental UA was developed on does not have this capability. A second alternative would be through the use of X-ON, X-OFF handshake signals. Most full duplex systems have this capability built into their I/O to allow a user to temporarily suspend output to the terminal. In the case of the UA, it would send an X-OFF byte to the CBMS whenever it was about to do file output and send the X-ON when it was ready again. Unfortunately, this simple handshake does not exist in the terminal I/O with one of the computers being used: the IBM mainframe, which operates in a half duplex mode.

There are two possible alternatives to emulate a similar technique with the IBM. The first would be to send an ASCII Break character down to the CBMS to stop output and a "resume typing" command when ready to start accepting output again. However this technique would still be prone to

losing characters that were sent before the CBMS acknowledged the break signal. The other alternative would be to use a public network such as Datapac which would allow a user to send X-OFF to a half duplex system. One of the disadvantages of this technique would be that the CBMS would then all have to be accessible via such public networks.

An ideal case would be if the mainframes communicated by a higher level protocol based on the ISO seven layer model[15]. This would allow the UA to achieve flow control through the standardized techniques as laid out in that protocol.

The approach taken in the experimental UA is to allocate sufficiently large buffers to accomodate most messages without any disk activity. In the event of a longer message it may be possible that part of the message may be lost. Typically this has proven to be only a few characters during the period of actual disk activity. This window of time for losing characters is also very dependent on the hardware configuration of the disk controller of the particular workstation that is being used and the baud rate at which the communications line is running.

In a standardised communications environment there would be a protocol to handle the communications between two autonomous computers. In addition to the above problem of

flow control, the protocol would be able to check for such problems as an error in transmission and to request a resend. Recovering from a mainframe crash would present a more serious problem. How would the UA determine that the CBMS is no longer responding and what sort of recovery mechanism would it try to employ? While it would be possible for the UA to detect these types of conditions it was deemed that such a robust system was beyond the scope of this thesis and therefore was not attempted. In the case of the present implementation, the UA would continue, simply sending its commands until it was finished.

5.4 Use of Other Tools:

As noted in the introduction one of the main aims of this thesis was to determine methods by which a software system may build on the functions that are already present. In the case of electronic mail for example, we wish to use the local editor rather than write one as a subsystem of the UA. In such a manner, it is possible for a user to have the best tools associated with his environment as well as developing more portable tools, such as the UA; we do not have to make so many assumptions about the hardware configuration, such as full screen capabilities, because these are handled by existing tools, such as the editor.

There are various mechanisms for achieving this type of interaction on larger computer systems. One such method on IBM mainframes is the use of "EXEC" files. These may communicate with programs through parameter passing and return arguments. The EXEC language has sufficient constructs to allow for testing and looping on conditions as well as variable assignment. In the case of other operating systems it is possible to call a program from within another and pass arguments to it through the use of such mechanisms as the FORK technique used in UNIX.

However, the CP/M system under which the experimental UA was developed does not have the mechanism to enable a user to call one program from another, or the multitasking primitives required for a pipe mechanism. The CP/M system does allow the user to define SUBMIT files which take parameters and these may be passed on to a program. There are, however, no mechanisms in CP/M for passing arguments back to the invoking file or for testing and looping constructs in the SUBMIT files.

One of the alternatives that could be utilized in this area is the use of a higher level supervisor program. Two such programs which were looked at were Microshell[10] and Supervyz[14]. However there are some inherent problems in the use of these. First, they require a significant amount of disk space. While this may not be a problem with larger

disk drives, the experimental UA only had 5 1/4" drives which also had to contain the operating system, system editor, application program and all the data files to run the UA with as well as the actual message files.

The memory space that these supervisor programs require presents a much more serious problem. A supervisor program is essentially a program that runs another program, and any space that these systems require decreases the amount of memory left for the application programs. While this may not be a problem in a 16 bit microcomputer system, it must be considered in the experimental UA which is using an 8 bit processor with a 64K memory limit.

Another approach to the problem is to have the user manually invoke the editor. The problems with this approach are numerous. The first is that the user must be aware of which file he must edit to change a message. Secondly, he would have to continually input his password in order to relog onto the UA. After getting back into the UA he would have to remember to input the command to deposit the file into the outbox. This approach would thus require a reliance on the user to input a series of commands in the right order and would also detract from trying to present an integrated system. He would still be aware of a separate program called an editor and would also have to know how to reinvoke the UA from the command interpreter.

The experimental UA uses the submit command to allow it to avoid most of the above problems. The compiler that has been used in its development allows for an application to return a command line back to the operating system. The UA makes use of this to invoke a submit file from itself. This submit file takes two parameters from the command line. The first one is the password that was used to log onto the UA. This allows it to be reinvoked without the user having to re-input the password. The second parameter is the command that is to be issued when the UA is re-entered.

The sequence of using the editor is the following. The user issues a command to the UA which requires use of the editor. The UA sets up the required temporary file and then invokes the submit file passing the two parameters which are required on returning. Then the editor is automatically invoked from the submit file. The user modifies the data file as desired. On exiting the editor, the operating system picks up the next line from the submit file. This line re-invokes the UA passing the required password and the instruction that will pick up the modified file and place it into the outbox.

5.5 Analysing Response:

As mentioned in section 4.2 there is a problem of how to tell when there are messages to be read on the external CBMS. When an interactive user logs onto a CBMS it will normally inform a user that there are messages waiting for him. However, this is not of use once the UA has already read one message. In the case of an interactive approach one possible operation is to do a read. From the response back the user either gets a message informing him there are no messages to be read or it will start reading the first one.

This was the first approach used in developing the UA. A read command was sent, then the results were compared with what the external CBMS was expected to generate when there were no more messages to be read. If the two did not match then the UA knew it was a message and put it into the inbox. If the two matched then the UA knew there were no more messages to be read.

Unfortunately, this technique did not work on The Source CBMS. The commands required by The Source when there are messages cause confusion when there are none. As a result, it is necessary to have the UA check for messages first, before attempting to read any. This has two implications for the performance of the UA. First, it must issue two sets of commands to read a message. The second

concerns checking for messages. On The Source, for example, there is no command within the CBMS to check for existence of messages. Therefore the UA must get out of the CBMS environment, issue the message check command and then re-enter the CBMS on The Source.

5.6 Non-printing Characters:

In testing out the experimental UA another problem with reading messages from external CBMS was uncovered. There is normally no problem in receiving messages that had originated from terminals with ordinary RS-232 hookups to the mainframes. However, when a full-screen editor is used to compose a message on a terminal with a high speed interface to the mainframe there are non-printing ASCII characters imbedded into the message. When the user attempts a local edit, the microcomputer's editor translates these to spurious characters. As a result, it was necessary to include a filter in the external read function so that any such characters were discarded.

5.7 Sending Copies:

The next problem that the UA has to deal with is the sending of copies. The UA has the capability to send messages to people on different CBMS. In a regular CBMS where all the recipients are on the same CBMS, sending a

copy is no problem. The user includes the copy names in the header of the message and the CBMS will insure that the other people obtain a copy. However there are problems when the recipients of a message are located on different CBMS. The CBMS will not allow a user to send a copy of a message without sending an original. This is a problem when the person to whom the message is addressed is located on a different mainframe.

A possible solution is to simply send a message to the copy recipient as though he was the original receiver. However, the recipient may not realize that the message he got is a copy and may respond differently than if he had been aware of it. Also, he would not be aware of who else had received the message and may forward it to another recipient. The experimental UA circumvents this problem by including a list of original recipient and those getting copies as part of the text of the message. This is carried out automatically by the UA, without specific action by the user.

5.8 Hardware Dependencies:

Some of the other problems uncovered while testing and developing the UA were in the hardware area. Some hardware dependencies are built into it. The UA presumes the use of an autodial modem, and accordingly does not have any built

in pauses for the user to manually dial the external mainframes and instruct the UA when to continue. In addition, the UA makes use of the known responses from the modem to determine if the mainframe computer has answered the call.

The second dependency uncovered is due to interactions with the Datapac network. For obvious reasons, the UA should not rely on the order that external CBMS are used. Therefore, when accessing an external mainframe the UA assumes the modem is not currently dialed into any CBMS. However, when a user logs off a mainframe which he has accessed via Datapac, the line is not automatically dropped. Nor is there a way to instruct Datapac to drop the telephone line through software commands[7]. The network holds the line for a minimum of two minutes on the assumption that the user will want to use the network to access the next mainframe. Therefore, the only way to drop the telephone line from the UA is to be able to control the DTR signal to the modem from software.

It should also be noted that the UA has already been transported from the original development microcomputer a Heath H-8, to the Matrox systems at the University. In so doing the only changes required were the new values of the port addresses in the global include file and new routines to obtain and drop the DTR line. Therefore it can be seen

that the UA itself is fairly independent of hardware requirements other than those stemming from its major function as a communications system. The major system dependency is that of the interaction between the C language and the computer's operating system, CP/M.

Chapter 6

Review of Achievements and Extensions

All the objectives discussed in Chapter 1 have been met and reported in this thesis. Due to the limitations of both time and the 8080, CP/M microcomputer that was used, there are a number of areas in which further development of the experimental UA may be desirable. These are discussed in section 6.1. In section 6.2 a number of broader extensions of the concepts developed in this thesis are reviewed. The chapter concludes in section 6.3 by briefly discussing the status of this work in the light of current developments.

6.1 Further Work on the Experimental UA:

While the UA in its present form is a useable system, some facilities remain which would make it more complete. The first addition would be a utility to update the CBMS commands file. An interactive, user friendly program would prompt the naive user for the required sequences to access the CBMS. Such a utility would not have to be very sophisticated since it would only have to prompt the user for the series of commands used by the UA. The utility would simply have to collect the responses from the user and append them onto the commands file.

Reducing the overhead involved in sending messages from the outbox would also be of benefit. The program currently scans the outbox and sequentially sends the messages from it. If the current message is to be sent to a different CBMS from the previous message, the UA will log off the present CBMS and onto the required one. If the outbox is in a random order there may be a lot of overhead in logging on and off external CBMS. An easy solution would be to sort the outbox by the recipient's CBMS before attempting to send the messages. An analogous problem occurs when sending copies of messages. The program simply goes through the list in sequential fashion. Ideally, the UA should sort on primary recipient and also keep track of copies of all messages. Then when the UA does log onto a CBMS all messages could be sent at once further reducing the system overhead. An extension would be to check for new messages while logged onto a CBMS for sending purposes.

The UA could also be improved by the use of special CBMS features. Some CBMS provide bulletin boards. Others allow a user to send registered messages. The UA now uses only those facilities required for the basic functionality of electronic messaging. Perhaps there should be some facility whereby features of specific CBMS might be used while disallowing their use on the other ones.

In the current implementation a recipient must be registered in the correspondents data file. This informs the UA which external CBMS to use and which external userid to send to. Some method to update this data file from the UA would be advantageous. Then when a message from a previously unknown correspondent was received, one could easily register him in order to generate a reply. This utility could be easily provided since it requires only that the UA accept user input and append it to the correspondents data file. Another alternative solution would be to specify which CBMS the UA is to use for a message rather than searching the list of known correspondents.

The UA would also be enhanced by the capability to file messages when they are sent. Currently the system will allow a user to file a message only when it is read. However the user might modify a message after it was saved but before it was sent. By saving the message transmitted, the user is assured of having a copy of the message as actually received.

Another facility that should be provided in the system is the capability to send a specific message from the outbox immediately. Occasions arise when an urgent message should be sent right away without having to send the rest of the outbox. In this case the user should be able to specify a

particular message, and the UA should automatically send that one as though it was the only one in the outbox.

6.2 Extensions of the Concepts Developed:

It should be noted that the concepts and approaches adopted in the development of the experimental UA are not restricted to it. Similar problems exist for other areas in the development of the office of the future.

One of the key roles of a manager is to condense and analyse information. To assist him, the same techniques that were used in the UA could be applied to a database system as well. Indeed there would be considerable overlap in the basic command sets as both systems would need to know how to access the various networks as well as how to log on and off the mainframe computers.

It should also be noted that while the UA only makes use of the system editor there is no inherent limitation on which editor it uses or how many external programs it could use. For example by simply modifying the submit file it would be possible to have all new messages run through a spelling package or word processing system. Messages would then return to the UA and be deposited into the outbox.

The experimental UA has shown that it is possible to develop a working system with the existing tools. However, ideally other facilities would be provided in the development of software toolboxes for the office of the future. Probably the most important would be the opportunity for one program to call another. The BDS C compiler provides part of this facility by allowing a user to exit a program while returning a string to the CP/M command interpreter.

However, there are certain limitations to this approach. Firstly, while a program may be invoked in this manner, there is no provision for the called program to return to the last state of the calling program. It also does not easily allow for the passing of parameters between the two programs. In the case of the UA, the data is passed in by means of a temporary file. One approach that theoretically would be better would be roughly based on the ADA rendezvous mechanism. In this approach, a program could register itself as a "server" for some function, for example an editor. The operating system would then suspend this program and return to the command interpreter. The application program would do a locate to determine if any server program had been registered. When requiring an edit function it would simply invoke the declared entry in the server program. Such a technique would allow for binding between programs to be done at run time instead of compile time. It would also allow a server to be used by more than

one application program. Thus it would become possible for system developers to build program modules such as editors, filers, graphic systems, etc. that could be invoked from any program. In order for applications to be able to use these, all that they would require is an interface description to the entries that they would like to use. This concept is essentially the one used in developing network operating systems as described in [15].

The next problem area that was looked at was the area of communications between two autonomous computers. This problem in general has been the source of exhaustive research concerning such areas as distributed networks and databases. However, such protocols are usually conceived as two co-operating programs on different computers. What is required in the case of the experimental UA is to be able to request the mainframe's operating system to set up an ISO application layer communications link to the CBMS. This would enable the UA to be able to have flow control on message data, check for transmission errors and recover from loss of communication lines. The intelligence and capabilities exist at the workstation end to be able to carry on a higher level protocol with a mainframe. Note also that such a facility on the microcomputer should be modeled as a server process, as mentioned previously, rather than being replicated for each software system that requires external communications.

The third area that was looked at was the capability to reconfigure systems without reprogramming. While the basic technique of having the UA data table driven is probably the best possible given the current state of the technology there are some facilities that could be improved on. The first is the I/O used to access the data files. The best solution here would be to use a database system to contain the data for external communications. This would allow for not only random access but better update and organization of different CBMS command facilities. Its use would also facilitate the use of common data with other programs as previously mentioned. Even if the use of a database was not feasible, the capability to do random access I/O, append and delete from files, and reset file pointers would greatly ease the disk activity of the current UA.

6.3 Conclusions:

This thesis was concerned with the development of a UA for an electronic message system for the office of the future. An electronic message system is one of the basic requirements in a software toolbox if the knowledge worker is to be able to communicate effectively.

The problems associated with developing this particular tool were threefold:

1. how to use the available tools from the UA application
2. how to communicate with other computers without high level protocols.
3. how to build a UA which may use new CBMS without reprogramming.

The first problem is the most fundamental one as far as tools for the office are concerned. This functionality has been provided in the experimental UA by the use of temporary disk files and the submit facility. Ideally what should be provided is the capability to suspend an application and to restore its state at a later time. Some operating systems such as UNIX provide preliminary steps in this direction. For example, under UNIX a program may start the editor as a new process and wait until the new process dies (from exiting the editor). This technique does not provide for return parameters and would still require the use of the intermediate file for editing.

The second area addressed concerns the problems involved in communicating with external mainframe computers. The UA was required to use existing CBMS without reprogramming them. This meant that the UA would have to emulate an interactive user to the CBMS. The experimental UA has done this through the use of the CBMS commands file and the timeout technique described in section 5.2. This problem

is already being addressed by the introduction of higher level protocols between computers, most notably the ISO seven layer model. All that is required is the mechanism to allow a workstation to use such models to communicate with the operating system of the mainframe.

The third problem looked at was how to reconfigure the UA to use additional CBMS in the user environment without reprogramming. This led to the approach of having the UA table driven from the CBMS command file. This problem will be alleviated for electronic messaging when more CBMS are developed using the UA - MTA interface. However, there still exist other CBMS that do not provide such an interface and other applications that are not yet standardized. Therefore the data table driven technique will probably have to be used for some time to come.

In conclusion it has been shown that a useable tool can be developed within existing facilities. The new standards that are evolving will aid in inter-computer communications and interfacing to different CBMS. However, it will be a period of years before these are generally available. In the meantime an approach such as the one that has been described here may provide some intermediate capabilities for the developing office of the future.

References

1. R. Abbott, "Letter from the Chair", ACM SIGOA, Newsletter, Vol. 1 No. 1, March 1980., pp. 1-2
2. P. Abrahams, "Electronic Islands", Communications of the ACM, Vol. 26 No. 4, April 1983, pp. 241-242
3. B. D. Software C Compiler V1.4 User's Guide, Lifeboat Associates, New York, New York, 1981.
4. CCITT Draft Recommendations on Message Handling, March, 1983.
5. COCOS User's Guide, Bell-Northern Research, Ottawa, Ontario, 1982.
6. CP/M User's Guide, Digital Research, Pacific Grove, California, 1978.
7. Datapac - Interactive Terminal Interface Specification and User's Manual, TransCanada Telephone System, Ottawa, Ontario, 1981
8. Envoy 100 - User's Guide, TransCanada Telephone System, Ottawa, Ontario, 1982.
9. J. Martin, Design and Strategy for Distributed Data Processing, New Jersey, Prentice-Hall, 1981.
10. Microshell User's Manual Version 2.0, New Generation Systems Incorporated, Reston, Va. 1982.
11. D. Redell, J. White, "Interconnecting Electronic Mail Systems", Computer, Vol. 16 No. 9, Sept. 1983, pp. 55-63
12. The Source User's Manual, Source Telecomputing Corporation, McLean, Virginia, 1981.
13. D. Stouder, "NARS Proposes Definition of Office Automation", ACM SIGOA Newsletter, Vol. 1 No. 1, March 1980., p. 15
14. Supervyz User's Guide and Reference Manual, Epic Computer Corporation, 1981.

15. A. Tanenbaum, Computer Networks, New Jersey, Prentice-Hall, 1981.
16. Uhlig, Faber, Bair, Office of the Future, New York, North-Holland, 1979.

Appendix A

Portability Issues

As with most major computer programs there are some machine dependencies of the UA. While steps were taken to minimize and isolate such areas, they inevitably occur as one develops a functioning program. However, in order to aid porting of the UA to another environment the problem areas are described below.

CP/M Dependencies:

The first major assumption of the UA is that the CP/M operating system is used. This has impacted its design in many ways. It affects how the UA is able to read parameters passed into it. The C language has defined two variables which allow a program to review the command line. This functionality has been adhered to in the BDS C implementation on CP/M. The use of CP/M also affects the format of file names that are used by the UA. As mentioned in the system description the UA is very much I/O oriented and thus any change in file structure (either in naming conventions or directory structure) could have a large impact on the design. A third facility of CP/M that is used is the submit file and the capability of the UA to give a command line back to the CI when exiting. This feature allows the system editor to be used by the UA.

USART Dependency:

The next hardware dependency of the UA concerns the I/O configuration. Due to the very nature of the application it is necessary for the microcomputer to be able to communicate with external mainframes. The first assumption made here is that the UA is using an Intel 8251 USART chip for serial communication. As mentioned previously it is also necessary for the UA to be able to drop the communications line to DATAPAC. Therefore it is also presumed that the DTR line on the serial link can be controlled through the use of software. The UA has also been developed on the premise that logging onto the mainframe systems should be totally automated. Therefore it assumes that it is connected to a RACAL-VADIC autodial modem. The type of modem is only important in that the UA must be aware of how to get the attention of the modem when it is about to initiate communications.

In addition to the assumption of the USART chip, the UA does not contain any code to initialize the baud rate. It is presumed either that the baud rate is fixed in hardware or, if the microcomputer has programmable baud rates, that they have already been initialized by system software. The UA also makes use of the BDS C Delay intrinsic function when communicating with the mainframes. As mentioned in their

manual this time delay is based on the assumption of an 8080, 2MHZ system[3]. However, this should not present a problem as any of these delay calls are based on data files. Therefore the delay parameters may be changed by the user without program modifications.

The other assumptions about the hardware environment concern memory available and number of disk drives. The UA was developed on a 64K memory size. Due to nesting levels of reading messages, it requires all the memory. If it was necessary to modify the UA to run on smaller memory size it could be achieved as follows. The BDS C compiler has various facilities that allow a program to overlay parts of memory. The UA is sufficiently modular to take advantage of this facility. For example, it is not necessary to contain the code for reading external messages while the UA is sending others out. It would be a simple matter to convert the main command loop to read in an overlay before branching to the required routine as is done now.

The last hardware requirement of the current UA is the use of two disk drives. This is not an inherent requirement of the UA itself but rather the result of not having enough room on the 5 1/4" diskettes to hold all the software as well as the required data files.

Appendix B

User Manual

This appendix explains the commands that are available to the user when using the UA.

A) Starting the UA:

The main program is called EMAIL and resides on the B disk. Therefore, to start the UA the user types B:EMAIL to the CP/M command prompt. It is important that the default disk be the A disk as the UA uses the submit facility and the operating system requires the submit file to be on the system disk.

The user also has the option of passing one or two parameters to the UA on the command line. The first parameter is the user's password. If it is not present the UA will prompt the user for it. The user has three chances to input the correct password before the UA will exit. If the user has given his password as the first parameter on the command line then he may also pass in a second parameter which the UA will use as the first command to execute. Thus a user may set up a builtin boot command which will automatically logon to the UA and retrieve the messages from the external CBMS.

B) List command (l)

The list command lists all the messages in the current inbox. This can be redefined from the default box by the new box command. The information that is printed is recipient's name, originator's name, copies, and subject line.

C) Send command (s)

This command sends the outbox to the external CBMS. This command leaves the user in the UA after the messages have been sent.

D) Exit command (e)

This command is essentially the same as the send command except that the user is returned to the CP M environment once the contents of the outbox have been sent.

E) Quit command (q)

This facility allows the user to return to the CP M environment without sending any messages from the outbox.

F) Transparent command (t)

This command allows the user to communicate directly with the external CBMS via the serial port. This is useful when a user wishes to use the external CBMS for other than electronic messaging. It is also of benefit in debugging the UA when there are problems with the inter-computer communications.

G) Correspondents command (u)

This command gives a list of all correspondents that are registered to receive messages from the UA. The first line is the internal name, the second line is the CBMS he is on, the third line is his userid on the external CBMS. The rest of the information is not used by the UA and may contain any information the user deems appropriate.

H) Create command (c)

The user invokes this facility when he wishes to create a new message. After the text has been input, the UA prompts for the envelope information. The new message is automatically deposited in the OUTBOX.

I) Read command (r)

This command allows a user to read a message from the INBOX. The message is specified by its number in the inbox (as given by the list facility). If the user inputs a carriage return when asked for a message number the UA reads the first message.

Once a message has been read the user is prompted for the action he wishes to take on it. The possible responses are:

- i) <cr> No action. Returns to main command prompt.
- ii) (f) Forward the message to another correspondent.
The UA allows the user to annotate the original message.
- iii) (a) Answer the message. The UA automatically generates the envelope for the reply. The user is put into the editor in order to compose his reply.
- iv) (s) Save the message in another file. This allows the user to sort his messages into meaningful files. If the specified file to save the message in does not exist, then the UA automatically creates it. If the file does exist then the message is appended to it.

J) Login command (i)

This command automatically logs a user into an external CBMS and then puts him into transparent mode. The user is prompted for the name of the system that he wishes to log into.

K) New inbox command (n)

This facility allows a user to redefine the inbox from the default file. Thus the user can scan and re-read messages that he has saved from previous reads. The user is prompted for the name of the new inbox.

L) External read (e)

This command allows a user to read all the messages that are waiting on a specified external CBMS. These messages are deposited into the (default) inbox. See also the read-all command. The user is prompted for the CBMS desired.

M) Read-all command (a)

This command reads all messages from the list of registered CBMS given in the SYSTEMS data file. This differs

from the external read facility in that the UA checks several external CBMS rather than the one specified.

N) Delete command (d)

This facility allows the user to delete messages from any message files. The user is prompted for the file name that he wishes to delete messages from. The UA then requires that the user give a list of message numbers to delete and terminates with a 0. It should be noted that it is up to the user to use this command in order to remove old messages from both the inbox and the outbox file.

O) Help facility (?, h, H, ~)

The UA uses an online help facility to aid a user. Whenever the UA requests input the user may obtain help by inputting ?, h, or H. This gives a message concerning the required information. In addition the user may input a ~ in order to chain up the sequence of commands that got him to the present level. For example if the UA is asking for the save file name and a ~ is input the UA gives the help information associated with the action request from read. Further ~ input continues to chain up until no further help can be given from the file.

Appendix C

Data File Formats

The UA requires several data files in order to operate. This appendix describes how these files are setup and how they can be modified. In the experimental UA all these files are ASCII text and therefore may be created with the system editor. Samples of these files are at the end of this appendix.

As mentioned in section 5.1 the I/O on the data files is done in a line oriented manner. It is important therefore that all lines, including the last one in the file, are terminated with a carriage return. Unless this is done, the BDS C I/O routines will not find the last line in the file.

1. CBMS commands (b:command.dat):

This file is used by the UA to allow it to engage in a dialogue with the external CBMS. The format of this file is split into a hierarchy. At the first level, the file is split into different CBMS sections. For each CBMS that the UA is to use there must be a corresponding section in this file. The first line of the section is its name. The end of the section is defined by an end of section character ('!', 21 HEX). For example, two CBMS, ENVOY and COCOS would be declared in the file as:

ENVOY

-- other data

COCOS

Within each CBMS section the file is further subdivided into each of the functions that are to be used with the external CBMS. As mentioned in section 4.1.2 these functions are logon, logout, scan, read mail, send, end send, analyse, and check end. These functions are delimited by an end of function character ('^', ^E HEX). Thus one part of the commands file may exist as:

```
envoy    -- section name
logon    -- function name
```



```
      .      -- end function  
      send    -- next function
```

```
      -- end section
```

Within this set of functions there are two distinct groupings: those that send input to the CBMS, and those that are used to analyse the responses from the external CBMS. The functions which are in the first category are logon, send, endsend, scan, read, and logout. The second category consists of checkend and analyse. Within the function groupings there may be several lines of data which are interpreted in different ways depending on the function name.

The first category functions simply contain data lines that should be sent as though they were interactively typed at a terminal. These are paired with timeout values which are used to determine when the command has been completed as described in section 5.2.

The one exception to this pairing rule is the telephone number that is given in the logon function. In order to provide some robustness, the UA knows how the modem will respond when a number is dialed. Thus the UA is able to tell if the external computer answered the phone call and will inform the user if it did not. Therefore no timeout value is needed for the telephone number.

Within this category of functions there are a few that require variable data. In the experimental UA the only one that falls into this category is the send function. However, a similar technique should be used for the logon function. Whenever a data value is to be inserted, the line in the commands file gives a parameter token. This token begins with the '&' character and then has a parameter number after it. The UA can tell from the context of which function it was executing, where it should obtain the required data from. The four variables that are defined for the send command are:

1. recipient
2. originator
3. carbon copies
4. subject

For example, send &1, would cause &1 to be replaced by the recipient's name for the current message before the line is

transmitted. As mentioned this same technique should be extended to the logon function to get userids and passwords from the account file. Currently the UA expects this data to be put directly into the commands file.

The other grouping of functions are those that inform the UA of how to analyse the responses from the CBMS. The first is the check end function. The data that is given here is a copy of what the UA would receive from the scan function if there are no more messages to be read. Thus a simple comparison of these datalines with the response from the scan function will let the UA know if there are any messages to be read from the CBMS.

The other function that must interpret the received data is the analyse function. This function is responsible for converting the received message from the external CBMS form to the internal one. As with the send command, the defined variable numbers are:

1. recipient's name
2. originator's name
3. carbon copies
4. subject

Each part of the analyse command is in 3 lines. The first gives the parameter number that this part is defining.

The next line gives the (space delimited) token that will appear at the beginning of the line. The last tells how many tokens make up this parameter and gives a list of their numbers (after the defining token is removed). If the number is 0 then the rest of the line is taken as the parameter. For example the analyse section for the ENVOY system is:

```

2          -- parameter 2
From:      -- beginning of data
1,1       -- one token, first one
1         -- parameter one
TO:
1,1
4         -- parameter four
Subj:
1,0       -- one token, rest of
          the line

```

Any none blank lines that are encountered between the start of the message and the last entry in the analyse command are copied as part of the message text. Once all the parameters defined in the analyse function have been satisfied then the rest of the message is copied to the text portion of the internal message.

2. Account file (b:account.dat):

This file is meant to contain all the necessary userids and passwords that would be required for logging onto the UA and the external CBMS. As such this information is sensitive and should be protected by encryption or some other means.

In the experimental UA the only information that is contained in this file is the password required when starting the UA. Thus the first line of this file contains the password that must be input when a user wished to logon to the UA.

3. Systems file (b:systems.dat):

This file contains the names of the external CBMS that should automatically be checked for messages by the "read all" command. The names of the CBMS given here correspond to the section names given in the CBMS commands file. That is a name here will enable the UA to find the correct section of the CBMS commands file to allow it to retrieve the desired messages.

4. Registered Correspondents (b:users.dat):

This file is used by the UA to determine which CBMS to access in order to send a message to a recipient. The

information in this file is also used for the translation of external names to internal ones and to provide the directory facility. Each of the correspondents records are delimited with an end of record symbol ('°', 7E HEX). The first line of the record is the internal name that the UA uses for this recipient. The second line is the external CBMS that is used to send messages to this user. The third line is the name that the external CBMS identifies the recipient by. The rest of the information is not used by the UA and is simply output to the terminal when using the directory command.

5. Edit command (a:tedit.sub):

This is the submit file that is executed whenever the UA wishes to invoke the system editor.

6. Message files (b:inbox.dat, b:outbox.dat, user files):

These files are the ones that contain the actual messages. The inbox and outbox are different only in that their names are known by the UA. Any incoming messages are stored in the inbox, and any created or forwarded messages are put in the outbox. The file is comprised of a series of concatenated messages. The first four lines of a message correspond to the four defined parameters as described for send. The rest of the message is the actual text. The end of

the message is delimited by an end of message character
(⁰, 7E HEX).

Sample Files:

CBMS commands file:

```
cocos
logon
567-9100
5
.
5
cocos datapac address
10
logon cocos userid
10
cocos password
5
cocos
.
send
5
compose
3
&1
3
&3
3
&4
5
input
.
endsend
4

5
file
4
send
.
scan
5
scan inbox unread
.
read
5 /
read
.
logout
10
bye
10
```


no
 10
 logout
 .
 checkend
 No such message(s) in INBOX
 .
 analyse
 1
 to:
 2,1,2
 2
 from:
 2,1,2
 4
 subject:
 1,0
 .
 !
 envoy
 logon
 567-9100
 5
 .
 5
 39400100
 8
 logon envoy userid
 10
 envoy password
 .
 send
 4
 compose
 4
 &1
 4
 &3
 4
 &4
 .
 endsend
 4
 .
 4
 yes
 .
 scan
 8
 scan
 .
 read
 5
 read

5
purge
10
exit
.
logout
10
bye
.
checkend
 No new messages.
.
analyse
2
From:
 1,1
 1
TO:
 1,1
 4
Subj:
 1,0
 .
 !
source
 logon
 567-9100
 5
 .
 5
p 1 3110 30100028
8
 logon source userid
 0
 source password
 12
 6
 .
send
 10
mail send
 10
 &1
 20
 &4
 .
endsend
 20
 .s
 20
quit
.
read
 20

2 2
OF / DE

mail read
20

6

6

6

quit

6

scan

60

mailbox

6

logout

60

logout

6

checkend

!lxYour Mailbox is empty at this time.

analyse

2

From:

1/1

1/1

1/1

1/1

4

Subject:

1/1

1/1

Account File:

RUSS

Systems file:

envoy
COCOS
source

Correspondents Data file:

RUSS MCDOWELL

cocos

R. McDowell

First line of information.

Second test line of info.

°

Roger Kaye

envoy

RogerKaye

Chair of Office Automation

Carleton University

Office: 231-2601

Home: 592-2598

°

R.McDowell

source

RD0156

Member of Scientific Staff

Bell-Northern Research

Office: 727-2139

°

M.R. McDowell

envoy

R.Mcdowell

Account for envoy use

°

delete. The list is terminated with a zero.
Message numbers are input one per line.

24

System is requesting the external system name that you wish to log into. The name given must be a valid entry in the commands data file.

25

System would like to know from which system you would like to read messages from. The system name must be as specified in the commands data file.

26

System is requesting data for the envelope.

91

Could not find the list of registered users.

92

Could not create temporary file for new message.

93

No inbox file to read the messages from.

94

Specified message not found in the inbox.

95

Could not create temporary file for read.

96

Unexpected end of file on reading message.

97

Could not open temporary file for forward.

98

Could not open temporary file for forward.

99

Problem in writing out the temporary message to forward.

100

Could not create file for answer.

101

Could not create temporary file for save.

102

Could not find message to save.

103

Could not create temporary file for reply.

delete. The list is terminated with a zero.
Message numbers are input one per line.

24

System is requesting the external system name that you wish to log into. The name given must be a valid entry in the commands data file.

25

System would like to know from which system you would like to read messages from. The system name must be as specified in the commands data file.

26

System is requesting data for the envelope.

91

Could not find the list of registered users.

92

Could not create temporary file for new message.

93

No inbox file to read the messages from.

94

Specified message not found in the inbox.

95

Could not create temporary file for read.

96

Unexpected end of file on reading message.

97

Could not open temporary file for forward.

98

Could not open temporary file for forward.

99

Problem in writing out the temporary message to forward.

100

Could not create file for answer.

101

Could not create temporary file for save.

102

Could not find message to save.

103

Could not create temporary file for reply.

•
104
Could not find message to reply.
•
105
Could not find message to append for reply.
•
106
Could not create file for forward command.
•
107
Could not find message for forward command.
•
108
Could not create file for create command.
•
109
Could not find message for create command.
•
110
Could not find file to delete messages from.
•
111
Could not find CBMS - command file.
•
112
Could not find requested system in the command file.
•
113
Could not find requested command in the command file.
•
114
Unexpected end of command file.
•
115
Could not create file for new inbox.
•
116
Could not find new message to put in inbox.
•
117
Unexpected end of new message while converting to
internal format.
•
118
Could not find users data file.
•
119
Unexpected end of file on CBMS - commands.
•
120
Unexpected end of file on CBMS - commands.
•
121

Unexpected end of file searching for command.

122

External computer system did not answer the modem.

Appendix D
System Structure Charts

The following system charts are based on the Yourdon structure chart symbols.

- PROGRAM MODULES

- LOOPING CONSTRUCT

- DECISION CONSTRUCT

- DATA

FIGURE D-1
TOP LEVEL STRUCTURE

FIGURE D-2
READ ALL CBMS
STRUCTURE

FIGURE D-3
PROCESS MSG STRUCTURE

FIGURE D-4
CONVERT MSG STRUCTURE

FIGURE D-5
LOGIN CBMS STRUCTURE

FIGURE D-6
READ MESSAGE STRUCTURE

FIGURE D-7
SEND STRUCTURE

FIGURE D-8
FORWARD MSG STRUCTURE

FIGURE D-9
ANSWER MSG STRUCTURE

Appendix E

System Code Files

The source files for the UA are contained in five different files. They are named file1, file2, file3, file4, file5.

```

/*****
/*
/*          LOCAL EMAIL SYSTEM          */
/*
/*      Author:  R. McDowell      (c)  1983.      */
/*
/*              (file1)              */
/*
*****/

/*****
/*
/*      This section contains the procedures that
/*      make up the major input, and help facilities
/*      of the local email system.
/*      They are the following procedures:
/*      main - main command loop of the program
/*      verifypassword - check users signon
/*                      password
/*      outputerror - output error message
/*      helpcmd - output help message to user
/*      getresponse - get a line of input from
/*                      user
/*      help - output help message to user
/*      badcmd - output message to user
/*
*****/
#include B:BDSCIO.H
#include b:variable.c

main(argc, argv)
int argc;
char *argv[];
{
char inputline[80];
int cont, exit;
/* Output title message */
puts("\n      E M A I L   O F F I C E   S Y S T E M");
puts("\n\n");
puts("              ( R. McDowell   (c) 1983. )");
puts("\n\n\n");
/* ensure that this is a valid user */
if (argc > 1)
    verifypassword(argv[1]);
else
    verifypassword("");

/* loop accepting commands until user exits */
cont = 1;
if (argc > 2)
    inputline[0] = tolower(*argv[2]);
else

```

```

        inputline[0] = ' ';
        inituart();
        strcpy(inbox, "b:inbox.dat");
        do
        {
            if (inputline[0] == ' ')
            {
                puts("\Command: ");
                get_response(2, inputline);
            }
            inputline[0] = tolower(inputline[0]);
            switch (inputline[0])
            {
                case 'l': listcmd();break;
                case 's': sendoutbox();break;
                case 'x': exit = 1; cont = 0;break;
                case 'q': exit = 0; cont = 0;break;
                case 't': transparent(0);break;
                case 'u': directorycmd();break;
                case 'c': createcmd(); break;
                case 'r': readcmd();break;
                case 'i': login();break;
                case 'n': puts("\nBox name: ");
                        get_response(21, inbox); break;
                case '1': appendreply(); break;
                case '2': appendforward(); break;
                case '3': appendcreate(); break;
                case 'e': ext_read(""); break;
                case 'a': read_all(); break;
                case 'd': delete_cmd(); break;

                default : badcmd(); break;
            }
            inputline[0] = ' ';
        }
        while (cont == 1);

        /* check for quick or normal exit */
        if (exit == 1)
            sendoutbox();
    }

    verifypassword(password)
    char password[]
    {
        int ioreturn, wrongcount, i;
        char iobuffer[BUFSIZ], inputstr[150], pass[80];
        /* first putpassword in local variable and convert
           to lower case */
        movmem(password, pass, strlen(password)+1);
        for (i = 0; i <= strlen(pass); i++)
            pass[i] = tolower(pass[i]);
        /* open the accounts data file */
        /* first line is system password */

```

```

ioreturn = fopen("B:ACCOUNT.DAT", iobuffer);
if (ioreturn == -1)
    {help(122); exit();}
else
    {
        ioreturn = fgets(inputstr, iobuffer);
        ioreturn = fclose(iobuffer);
        /* loop, allowing 3 errors */
        wrongcount = 1;
        strcat(pass, "\n");
        while ((strcmp(pass, inputstr) != 0) && (wrongcount < 4))
            {
                ++wrongcount;
                puts("Password: ");
                get_response(1, pass);
                strcat(pass, "\n");
            }

        /* check for failure */
        if (wrongcount > 3)
            {
                help(122);
                exit();
            }

        /* save user password for future use in invoking
           editor and returning */
        movmem(pass, userpass, strlen(pass));
        userpass[strlen(pass)-1] = 0;
    }
}

```

```

get_response(function, buffer)
char buffer[];
int function;
{
    int num;
    char input_line[150];
    num = function;
    while (1)
        {
            gets(input_line);
            /* now check for help request */
            if (input_line[0] == '?')
                help(function);
            else if (strcmp(input_line, "help") == 0)
                help(function);
            else if (strcmp(input_line, "HELP") == 0)
                help(function);
            else if (input_line[0] == '-')
                {
                    num = num / 10;
                    help(num);
                }
            else

```



```

        {
            strcpy(buffer, input_line);
            return;
        }
    }

help(function)
int function;
{
    char iobuf[BUFSIZ], input_line[150];
    int ioreturn, num, present_msg;
    /* open the help file to read message */
    ioreturn = fopen("B:HELP.MSG", iobuf);
    if (ioreturn == -1)
    {
        printf("\nFunction code: %u",
            function);
        return;
    }
    else
    {
        /* scan input file for requested message */
        while (1)
        {
            ioreturn = fgets(input_line, iobuf);
            num = atoi(input_line);
            if ((num == function) || (ioreturn == 0))
                break;
            else
            {
                /* skip to next message */
                while (strcmp(input_line, "" "\n") != 0)
                    fgets(input_line, iobuf);
            }
        }
        /* check for ioreturn or msg found */
        if (ioreturn == 0)
        {
            printf("\nNo help available for function code %u",
                function);
        }
        else
        {
            /* have found message, loop printing contents */
            while (1)
            {
                fgets(input_line, iobuf);
                if (strcmp(input_line, "" "\n") == 0)
                    break;
                puts(input_line);
            }
        }
        fclose(iobuf);
    }
}

```

```

    }
}

badcmd()
{
    puts("\nInput was a bad command.");
    help(2);
}

/*****
/*
/* This section deals with hardware dependent features. */
/* The routine that are involved here are: */
/*     read_wait - read characters into buffer till pause */
/*     pause_wait - wait for pause limit without reading */
/*     kick_modem - get modem reading for commands */
/*
*****/

read_wait(outbuf, pause_time)
char outbuf[];
int pause_time;
{
    int count, index, a;
    int loop_count;
    char outline[150], inputchar;

    loop_count = pause_time;
    while (loop_count > 0)
    {
        --loop_count;
        /* adjust pause time to seconds */
        count = 3000;
        index = 0;
        while (count != 0)
        {
            /* reset loop count for every character read */
            --count;
            if (serrdy())
            {
                /* character on serial port */
                inputchar = serin();
                if (inputchar != 0 && inputchar != 19 &&
                    inputchar != 127)
                {
                    outline[index] = inputchar;
                    if (inputchar == CR)
                    {
                        outline[index] = 0;

                        /* end of line, output to
                           temporary file */
                        fputs(outline, outbuf);
                        index = 0;
                    }
                }
            }
        }
    }
}

```

```

        }
        else
            index++;
    }
    count = 3000; loop_count = pause_time;
}

}

}

pause_wait(timeout)
int timeout;
{
    /* This routine will return to caller when the host port
       has not received any more characters for a
       time specified */
    int a, count;

    count = timeout;
    while (count--)
    { if (serrdy())
      { count = timeout; a = serin(); }
      sleep(10);
    };
}

kickmodem(line)
char line[];
{
#define CTRL_E 5
#define CR 13
int a;
int index;
char buffer[25];
/* This routine will get the attention
   the autodial modem */
/* requires ctrl-e */
sleep(10);
soutp(CTRL_E);
soutp(CR);

/* wait until '*', indicates modem is ready */
/* while ((a=serin()) != '*'); */
pause_wait(5);

/* modem is now ready */
soutstr("d\n");
sleep(30);
soutstr(line);
sleep(30);
soutstr("\n");
index = 0;
while ((a=serin()) != CR)

```

```
        buffer[index++] = a;
buffer[index] = 0;
if (strcmp(buffer, "\nDIALING: ON LINE") != 0)
    { help(122); exit(); }
sleep(50);
}
```

```

/*****
/*
/*          LOCAL EMAIL SYSTEM          */
/*
/*          Author:  R. McDowell          (c)  1983.
/*
*****/

/*****
/*
/*          This section deals with hardware
/*          dependent features of the system. The
/*          modules in this section are:
/*          transparent - puts the system in
/*                      talk thru mode
/*          inituart - initializes the USART
/*                   chip for transmission
/*          soutp - outputs a character out
/*                the alternate port
/*          soutstr - outputs a string out the
/*                 alternate port
/*          serin - reads a character from the
/*                serial port
/*          pause - causes the system to pause
/*                for a specified time
/*          serrdy - tells if there is a
/*                 character on serial port
/*
*****/

#include B:BDSCIO.H
#include b:variable.c

transparent(doinit)
int doinit;
{

int a, al;
/* initialize the usart */
if (doinit == 1) inituart();
get_line();
/* loop accepting characters until end of session
   is input */
do
{ /* check for host character */
  if (serrdy())
    {a=serin(); putchar(a);}
  else if (kbhit() != 0)
    {al=getchar();
      if (al == '&') filetransfer();
      else soutp(al);
    };
}
while (al != '\0');

```

```

drop_line();
}

inituart()
{
    outp(HSTATUS, 129);
    outp(HSTATUS, 64);
    outp(HSTATUS, 122);
    outp(HSTATUS, 23);
}

soutp(datachr)
int datachr;
{
    /* check for the infamous newline character */
    if (datachr == 10) datachr = 13;

    while ((inp(HSTATUS) & READYTX) == 0);
    outp(HDATA, datachr);
}

filetransfer()
{
    char buff2[15];
    char filename[15], iobuf[BUFSIZ];
    char inputstr[100];
    int ioreturn;
    /* get the name of the file to be opened */
    puts("\nEnter CP/M file name: ");
    gets(filename);
    ioreturn = fopen(filename, iobuf);
    if (ioreturn == -1)
        {puts("\nError in file open");}
    else
    {
        puts("\nEnter IBM file name: ");
        gets(filename);
        soutstr("edit ");
        soutstr(filename);
        soutstr("\n");
        pause();
        soutstr("case m\n");
        pause();
        soutstr("input\n");
        pause();
        /* now loop sending up lines till end
           of file encountered */
        while(fgets(inputstr,iobuf)!= 0)
        {
            if (strcmp(inputstr,"\n") == 0)
                soutstr(" \n");
            else soutstr(inputstr);
            /* wait for x-on from 370 */
            pause();
        }
    }
}

```

```

    }
    /* finished file, close and save on 370 */
    soutstr("\n");
    pause();
    soutstr("file\n");
    ioreturn = fclose(&iobuf);
}

soutstr(buffer)
char *buffer;
{
    while (*buffer != '\0')
        {soutp(*buffer);
        buffer++;
        };
}

serin ()
{
    int a;
    while (((a = inp(HSTATUS)) & READYRX) != 2);
    a = inp(HDATA);
    return(a);
}

pause()
{
    int charin;
    while ((charin=serin()) != XON);
}

serrdy()
{
    int a;
    a = inp(HSTATUS);
    if ((a & READYRX) != 0) return(1);
    else return(0);
}

drop_line()
{
    outp(DTRPORT, 0);
}

get_line()
{
    outp(DTRPORT, 1);
}

```

```

/*****
/*
/*          LOCAL EMAIL SYSTEM
/*
/*      Author:  R. McDowell      (c)  1983.
/*
*****/

```

```

/*****
/*
/*      This section contains the functions
/*      that are used to do most of the local
/*      work of the system. These functions
/*      are:
/*          listcmd - implements the list
/*                   feature
/*          directorycmd - implements the
/*                   directory feature
/*          createcmd - implements the create
/*                   feature
/*          readcmd - implements the read
/*                   feature
/*          forwardcmd - implements the
/*                   forward feature
/*          answercmd - implements the answer
/*                   command
/*          savecmd - implements the save
/*                   feature
/*          appendreply - puts reply in outbox
/*          appendcreate - puts new mail into
/*                   outbox
/*          createenvelope - creates envelope
/*                   info for messages
/*          deletcmd - allows user to delete
/*                   messages from files
/*
*****/

```

```

#include B:BDSCIO.H
#include b:variable.c

```

```

listcmd()
{
    int ioreturn, msgnum;
    char inputstr[150], iniobuf[BUFSIZ];

    /* this procedure lists all the messages in
       the users inbox */
    ioreturn = fopen(inbox, iniobuf);
    if (ioreturn == -1)
        help(93);
    else
    {
        msgnum = 1;
    }
}

```



```

while (fgets(inputstr, iniobuf) != 0)
{ /* output message number, author,
   cc and subject */
  printf("%s%u", "\nMessage number: ",
        msgnum);
  puts("\nTo: ");
  puts(inputstr);
  /* get the author information */
  ioreturn = fgets(inputstr, iniobuf);
  puts("From: ");
  puts(inputstr);
  ioreturn = fgets(inputstr, iniobuf);
  puts("cc: ");
  puts(inputstr);
  ioreturn = fgets(inputstr, iniobuf);
  puts("Subject: ");
  puts(inputstr);
  /* now loop over the text of the message */
  do
  { ioreturn = fgets(inputstr, iniobuf);
    if (ioreturn == -1) break;}
  while (strcmp(inputstr, "" "\n") != 0);
  ++msgnum;
}

/* now close the file and return */
ioreturn = fclose(&iniobuf);
}

directorycmd()
{
/* this procedure implements the directory command
   for the system. The purpose of the command is
   to list registered users of the system and any
   other information that may be stored about them. */
int ioreturn;
char inputstr[150], iobuffer[BUFSIZ];

/* Try to open the user list data file */
ioreturn = fopen("B:USERS.DAT", iobuffer);
if (ioreturn == -1)
{ /* no list of known users found,
   output error message */
  help(91);
}
else
{ /* loop thru file outputting any
   user information */
  while (fgets(inputstr, iobuffer) != 0)
  { if (strcmp(inputstr, "" "\n") == 0)
    puts("\n");
    else
    puts(inputstr);
  }
}
}

```

```

    };
    /* now close the file and return */
    ioreturn = fclose(&iobuffer);
}

createcmd()
{
    /* This command will create a temporary file
       for the user to put his message into */
    char outbuf[BUFSIZ];

    if (fcreat("B:TEMPIN.DAT", outbuf) == -1)
        { help(92); exit(); }
    fputs("Dear: ", outbuf);
    putc(26, outbuf);
    fflush(outbuf);
    fclose(outbuf);
    execl("submit", "tedit", userpass, "3", 0);
    exit();
}

readcmd()
{
    int ioreturn, messageno, presentno, noint;
    char inputstr[150], iniobuf[BUFSIZ], outbuf[BUFSIZ];

    /* check for message number */
    puts("\nMessage Number: ");
    noint = scanf("%u", &messageno);
    if (noint == 0) messageno = 1;
    /* open the inbox file to read message */
    ioreturn = fopen(inbox, iniobuf);
    if (ioreturn == -1)
        help(93);
    else
    {
        presentno = 1;
        /* scan input file for requested message */
        while (presentno != messageno)
        {
            ioreturn = fgets(inputstr, iniobuf);
            if (ioreturn == -1)
                { fclose(iniobuf); help(94); break; }
            else
                if (strcmp(inputstr, "" "\n") == 0)
                { /* end of message,
                   increment count */
                    presentno = ++presentno; };
        };
        /* check if error or actually found message */
        if (presentno == messageno)
        {
            /* open a file for temporary storage */
            ioreturn = fcreat("B:TEMPOUT.DAT", outbuf);

```

```

    if (ioreturn == -1)
        {fclose(iniobuf); help(95); exit();}
    /* loop printing this message */
    while (1)
    {
        ioreturn = fgets(inputstr, iniobuf);
        if (ioreturn == -1)
            {fclose(iniobuf); fclose(outbuf);
             help(96); break;}
        else
        {
            /* check for end of record */
            if (strcmp(inputstr, "\n") == 0)
                break;
            else
            {
                ioreturn = fputs(inputstr, outbuf);
                puts(inputstr);
            }
        }

        /* flush and close temporary file */
        fputs("\n", outbuf);
        ioreturn = putc(26, outbuf);
        ioreturn = fflush(outbuf);
        ioreturn = fclose(outbuf);
    }

    /* close the file */
    ioreturn = fclose(iniobuf);
    /* ask the user if he wishes to reply or forward */
    puts("\nAction: ");
    get_response(220, inputstr);
    switch (inputstr[0])
    {
        case 'f' : forwardcmd(); break;
        case 'a' : answercmd(); break;
        case 's' : savecmd(); break;
    }
}

forwardcmd()
{
    /* This command will forward a message */
    char inbuf[BUFSIZ], outbuf[BUFSIZ];
    char inputline[150];
    int ioreturn;

    /* Open the temporary file to get the message */
    if (fopen("B:TEMPOUT.DAT", inbuf) == -1)
        {help(97); exit();}

    /* Now create file which user will edit */
    if (fcreat("B:TEMPIN.DAT", outbuf) == -1)

```

```

        {help(98); fclose(inbuf); exit();}
fgets(inputline, inbuf);
fputs("Original To: ", outbuf);
fputs(inputline, outbuf);
fgets(inputline, inbuf);
fputs("Original From: ", outbuf);
fputs(inputline, outbuf);
fgets(inputline, inbuf);
fputs("Original cc: ", outbuf);
fputs(inputline, outbuf);
fgets(inputline, inbuf);
fputs("Original subject: ", outbuf);
fputs(inputline, outbuf);
fputs("Forwarded message:\n", outbuf);
/* Now loop copying contents of file */
while (1)
{
    fgets(inputline, inbuf);
    if (strcmp(inputline, "\n") == 0)
        break;
    if (fputs(inputline, outbuf) == -1)
        {help(99); fclose(inbuf); exit();}
}
fputs("**** Note ****\n", outbuf);
/* Now close files and inform user to edit file */
putc(26, outbuf);
fflush(outbuf);
fclose(outbuf);
fclose(inbuf);
execl("submit", "tedit", userpass, "2", 0);
exit();
}

answercmd()
{
    /* This routine will generate the envelope for
       a reply to a received message. */
    char inbuf[BUFSIZ], outbuf[BUFSIZ];
    char inline1[150], inline2[150];
    int ioreturn;

    /* Create file for user to edit */
    if (fcreat("B:TEMPIN.DAT", outbuf) == -1)
        {help(100); exit();}
    fputs("Answer: \n", outbuf);
    /* now close files, tell user to edit file */
    putc(26, outbuf);
    fflush(outbuf);
    fclose(outbuf);
    execl("submit", "tedit", userpass, "1", 0);
    exit();
}

```

```

savecmd()
{
    /* This command will append a message to a file. */
    char inbuf[BUFSIZ], outbuf[BUFSIZ];
    char inputline[150];
    char filename[15];

    puts("\nfilename: ");
    get_response(2200, filename);
    /* First check if the file exists */
    if (fopen(filename, inbuf) == -1)
        /* File does not exist, simply rename file
           to desired one */
        rename("B:TEMPOUT.DAT", filename);
    else
    { /* File exists, copy to a temporary one,
       then append */
        if (fcreat("B:TEMPSAVE.DAT", outbuf) == -1)
            { help(101); fclose(inbuf); exit(); }
        while (fgets(inputline, inbuf) != 0)
            fputs(inputline, outbuf);
        /* now copy message into file */
        fclose(inbuf);
        if (fopen("B:TEMPOUT.DAT", inbuf) == -1)
            { help(102); exit(); }
        while (fgets(inputline, inbuf) != 0)
            fputs(inputline, outbuf);
        /* Now close all files and rename the
           temporary one */
        putc(26, outbuf);
        fflush(outbuf);
        fclose(outbuf);
        fclose(inbuf);
        /* Delete file first */
        unlink(filename);
        rename("B:TEMPSAVE.DAT", filename);
    }
}

/*****/

appendreply()
{
    /* This function is used to append the temporary
       reply file to the outbox. This differs from
       the forward and create functions in that
       the envelope is automatically generated. */

    char outbuf[BUFSIZ], inbuf[BUFSIZ];
    char inline1[150], inline2[150];
    int valid_outbox;

```

```

/* First create a new file to copy outbox into */
if (fcreat("B:TEMPBOX.DAT", outbuf) == -1)
    { help(103); exit(); }
valid_outbox = 0;
/* Now check for outbox file to be copied */
if (fopen("B:OUTBOX.DAT", inbuf) != -1)
    { /* Outbox exists, copy contents to new box. */
        valid_outbox = 1;
        while (fgets(inline1, inbuf) != 0)
            fputs(inline1, outbuf);
        fclose(inbuf);
    }
/* Now generate envelope for reply */
if (fopen("B:TEMPOUT.DAT", inbuf) == -1)
    { help(104); exit(); }
fgets(inline1, inbuf);
fgets(inline2, inbuf);
fputs(inline2, outbuf);
fputs(inline1, outbuf);
/* Always send cc's for now */
fgets(inline1, inbuf);
fputs(inline1, outbuf);
/* now the subject */
fgets(inline1, inbuf);
fputs("Re: ", outbuf);
fputs(inline1, outbuf);
/* Now close file and then copy response */
fclose(inbuf);
if (fopen("B:TEMPIN.DAT", inbuf) == -1)
    { help(105); exit(); }
while (fgets(inline1, inbuf) != 0)
    fputs(inline1, outbuf);

/* Now have new outbox, close files and
   rename if necessary */
fputs("\n", outbuf);
putc(26, outbuf);
fflush(outbuf);
fclose(outbuf);
fclose(inbuf);
if (valid_outbox)
    unlink("B:OUTBOX.DAT");
rename("B:TEMPBOX.DAT", "B:OUTBOX.DAT");
}

appendforward()
{
/* This function will append the temporary
   file onto the outbox */
char outbuf[BUFSIZ], inbuf[BUFSIZ];
char inputline[150];

```

```

/* create new file, and copy outbox into it */
if (fcreat("B:TEMPBOX.DAT", outbuf) == -1)
    { help(106); exit(); }
if (fopen("B:OUTBOX.DAT", inbuf) != -1)
    { while (fgets(inputline, inbuf) != 0)
        fputs(inputline, outbuf);
      fclose(inbuf);
    }

/* Now put envelope into outbox */
create_envelope(outbuf);

/* Now copy forwarded message into outbox */
if (fopen("B:TEMPIN.DAT", inbuf) == -1)
    { help(107); exit(); }
while (fgets(inputline, inbuf) != 0)
    fputs(inputline, outbuf);
fputs("\n", outbuf);
putc(26, outbuf);
fflush(outbuf);
fclose(outbuf);
fclose(inbuf);
/* Now delete outbox and rename temporary one. */
unlink("B:OUTBOX.DAT");
rename("B:TEMPBOX.DAT", "B:OUTBOX.DAT");
}

appendcreate()
{
/* This function will append the temporary
   file onto the outbox */

char outbuf[BUFSIZ], inbuf[BUFSIZ];
char inputline[150];

/* create new file, and copy outbox into it */
if (fcreat("B:TEMPBOX.DAT", outbuf) == -1)
    { help(108); exit(); }
if (fopen("B:OUTBOX.DAT", inbuf) != -1)
    { while (fgets(inputline, inbuf) != 0)
        fputs(inputline, outbuf);
      fclose(inbuf);
    }

/* Now put envelope into outbox */
create_envelope(outbuf);

/* Now copy forwarded message into outbox */
if (fopen("B:TEMPIN.DAT", inbuf) == -1)
    { help(109); exit(); }
while (fgets(inputline, inbuf) != 0)
    fputs(inputline, outbuf);
fputs("\n", outbuf);
putc(26, outbuf);

```

```

fflush(outbuf);
fclose(outbuf);
fclose(inbuf);
/* Now delete outbox and rename temporary one. */
unlink("B:OUTBOX.DAT");
rename("B:TEMPBOX.DAT", "B:OUTBOX.DAT");
}

create_envelope(buf_ptr)
char buf_ptr[];
{
/* This routine will create an envelope for a
new message to be put into the outbox */

char inputline[150];

puts("\nTo: ");
gets(inputline);
fputs(inputline, (buf_ptr));
fputs("\n", buf_ptr);
puts("\nFrom: ");
gets(inputline);
fputs(inputline, (buf_ptr));
fputs("\n", buf_ptr);
puts("\ncc: ");
gets(inputline);
fputs(inputline, (buf_ptr));
fputs("\n", buf_ptr);
puts("\nSubject: ");
gets(inputline);
fputs(inputline, (buf_ptr));
fputs("\n", buf_ptr);
}

delete_cmd()
{
char boxname[18], inbuf[BUFSIZ], outbuf[BUFSIZ];
char inputstr[150];
int io_error, del_index, num_out, inum;
int outnum, msg_no, del_array[50];

/* this command allows a user to remove
messages from a file */

/* get the file name from the user */
puts("\nBox name: ");
get_response(23, boxname);
io_error = fopen(boxname, inbuf);
if(io_error == -1)
{help(110); exit();}

/* valid boxname, now create temporary file to
hold messages. */
fcreat("b:tempout.dat", outbuf);

```



```

/* now get message numbers to delete */
del_index = 0;
do
{
    puts("\nMsg. No.: ");
    get_response(231, inputstr);
    inum = atoi(inputstr);
    del_array[del_index++] = inum;
}
while (inum != 0);

/* now copy all the rest of the messages to the
   temporary file */
outnum = 1;
del_index = 0;
num_out = 0;

/* loop copying box to temporary file */
/* do not copy any messages whose number
   appears in list */

msg_no = del_array[del_index];
while (msg_no != -1)
{
    while (outnum != msg_no)
    {
        io_error = fgets(inputstr, inbuf);
        if(io_error == 0)
            break;
        else
        {
            fputs(inputstr, outbuf);
            /* check for end of msg */
            if (strcmp(inputstr, "\n") == 0)
            {
                outnum++;
                num_out++;
            }
        }
    }

    /* now doing msg in delete array, skip over */
    if (outnum == msg_no)
    {
        do
        {
            fgets(inputstr, inbuf);
            while(strcmp(inputstr, "\n") != 0);
            del_index++;
            msg_no = del_array[del_index];
            outnum++;
        }
        else
            break;
    }
}

```

```
    }  
    /* have now read the entire file,  
       check if outbox empty */  
    fclose(inbuf);  
    unlink(boxname);  
    /* if any messages saved, rename temporary  
       file to boxname */  
    putc(26, outbuf);  
    fflush(outbuf);  
    fclose(outbuf);  
    if (num_out == 0)  
        unlink("b:tempout.dat");  
    else  
        rename("b:tempout.dat", boxname);  
}
```

```

/*****
/*
/*          LOCAL EMAIL SYSTEM
/*
/*      Author:  R. McDowell      (c)  1983.
/*
/*
*****/

/*****
/*
/*      This section contains some of the
/*      procedures that are used to communicate
/*      with the external CBMS. The names
/*      are:
/*          login - login to an external CBMS
/*          sendcmd - send a message to an
/*                   external CBMS
/*          read_all - read all messages from
/*                   an external CBMS
/*          ext_read - read a message from an
/*                   external CBMS
/*          read2_mail - read mail
/*          process_message - check if there
/*                          is a message and process it
/*
*****/

#include B:BDSCIO.H.
#include b:variable.c

login(sys_name)
char *sys_name;
{
    int ioreturn, whichret;
    char inputstr[150], iniobuf[BUFSIZ];
    char system_name[50], pause_str[50], inputline[150];
    int error_flag, logingon, pause_time;

    /* get user information data */
    ioreturn = fopen("b:command.dat", iniobuf);
    if ( ioreturn == -1 )
        help(111);
    else
    {
        /* tell the user this will take time */
        puts("\nPlease be patient, this will take ");
        puts("a couple of minutes.\n");
        /* initialize the usart to talk to modem */
        inituart();
        get_line();
        whichret = 1;
        if (strcmp(sys_name, "") == 0)
        {
            whichret = 0;

```

```

        puts("\nSystem name: ");
        get_response(24, system_name);
        strcat(system_name, "\n");
    }
    else
        movmem(sys_name, system_name,
            (strlen(sys_name)+1));
    /* Obtain system section of commands file */
    error_flag = get_sys(system_name, iniobuf);
    if (error_flag)
    { /* Cannot find given system,
        output message */
        help(112);
    }
    else
    {
        error_flag = get_cmd("logon\n", iniobuf);
        if (error_flag)
            help(113);
        else
        {
            /* Now have system and command */
            /* get autodial modem to ring line */
            ioreturn = fgets(inputline, iniobuf);
            if(ioreturn == 0)
                { help(123); exit();}
            kickmodem(inputline);
            logingon = 1;
            do
            { /* Loop reading wait time
                and commands */
                ioreturn = fgets(pause_str, iniobuf);
                if(ioreturn == 0)
                    {help(114); break;}
                if(strcmp(pause_str, "o\n") == 0)
                    {logingon = 0;}
            }
            else
            { pause_time = atoi(pause_str);
              ioreturn = fgets(inputline,
                              iniobuf);
              soutstr(inputline);
              pause_wait(pause_time);
            }
        }
        while (logingon);
        fclose(iniobuf);
        if (whichret == 0) transparent(0);
    }
}

}

sendcmd()
{

```

```

    puts("\nSend command not implemented.");
}

read_all()
{
    /* This procedure will logon to all registered
       CBMS and retrieve the mail from them */
    char localbuf[BUFSIZ], system_name[15];
    int io_error;

    /* open the file with the system names */
    io_error = fopen("b:systems.dat", localbuf);
    if (io_error != -1)
    { /* have found data file,
       loop through all systems */
        do
        {
            io_error = fgets(system_name, localbuf);
            if (io_error != 0)
            { /* valid system name,
               get mail from it */
                ext_read(system_name);
            }
        }
        while (io_error != 0);
        /* have now read all system names from file,
           close data file */
        fclose(localbuf);
    }
}

ext_read(sys_name)
char *sys_name;
{
    /* this routine will read all the messages from
       a given system */
    char to_system[30];
    if (strcmp(sys_name, "") == 0)
    {
        puts("\nSystem name: ");
        gets(to_system);
        strcat(to_system, "\n");
    }
    else
        movmem(sys_name, to_system, strlen(sys_name)+1);

    /* now loop telling routine to read messages while
       not at end of messages */
    login(to_system);
    while(read2_mail(to_system));
    logout(to_system);
}

```

```

read2_mail(tosystem)
char tosystem[];
{
/* This command will read a message from an
external email system */
char inputline[150], pause_str[15];
int reading, pause_time, ioreturn;
/* open the command file */
fopen("b:command.dat", iniobuf);
get_sys(tosystem, iniobuf);
get_cmd/"scan\n", iniobuf);

/* delete the temporary file */
fclose("b:tempin.dat", outbuf);

/* now send commands to read the message */
reading = 1;
do
{
/* loop sending read commands */
fgets(pause_str, iniobuf);
if (strcmp(pause_str, ""\n") == 0)
    reading = 0;
else
{
    pause_time = atoi(pause_str);
    fgets(inputline, iniobuf);
    soutstr(inputline);
    read_wait(outbuf, pause_time);
}
}
while(reading);
/* now close the files */
fclose(iniobuf);
ioreturn = fputs("\n", outbuf);
ioreturn = putc(26, outbuf);
ioreturn = fflush(outbuf);
ioreturn = fclose(outbuf);
return(process_msg(tosystem));
}

process_msg(to_system)
char to_system[];
{
/* this routine will check if the message read
indicates end of messages or valid message.
If valid message then reformat it
to internal format and put in inbox */
char input_str[150], eomsg_str[150], in_line[150];
char pause_str[15];
int pause_time;
int valid_inbox, comp, checking, reading, eomsg;
fopen("b:tempin.dat", inbuf);
fopen("b:command.dat", iniobuf);

```

```

get_sys(to_system, iniobuf);
get_cmd("checkend\n", iniobuf);

/* now loop checking every line against an end
   of message output */
checking = 1;
eomsg = 1;
do
{
    fgets(eomsg_str, iniobuf);
    if(strcmp(eomsg_str, ""\n") == 0)
        checking = 0;
    else
    {
        do
        {
            fgets(input_str, inbuf);
            while (scan_string(input_str) == 0);
            comp = strcmp(input_str, eomsg_str);
            if (comp != 0)
            {
                /* strings not identical,
                   process as message */
                checking = 0;
                eomsg = 0;
            }
        }
    }
} while (checking);

/* now see if it was eom or valid message */
fclose(inbuf);
fclose(iniobuf);
if (!eomsg)
{
    /* delete the temporary file */
    fopen("b:command.dat", iniobuf);
    get_sys(to_system, iniobuf);
    get_cmd("read\n", iniobuf);
    fcreat("b:tempout.dat", outbuf);

    /* now send commands to read the message */
    reading = 1;
    do
    {
        /* loop sending read commands */
        fgets(pause_str, iniobuf);
        if (strcmp(pause_str, ""\n") == 0)
            reading = 0;
        else
        {
            pause_time = atoi(pause_str);
            fgets(in_line, iniobuf);
            soutstr(in_line);
            read_wait(outbuf, pause_time);
        }
    }
}

```

```

    }
    while(reading);
    /* now close the files */
    fclose(iniobuf);
    putc(26, outbuf);
    fflush(outbuf);
    fclose(outbuf);
    /* reformat the input file to internal format */
    convert(to_system);
    /* copy temp file into inbox */
    /* first copy existing inbox */
    if (fcreat("b:tempbox.dat", outbuf) == -1)
        {help(115); exit();}
    valid_inbox = 0;
    /* now check for inbox to be copied */
    if(fopen("b:inbox.dat", inbuf) != -1)
        { /* inbox exists, copy contents
           to new box */
          valid_inbox = 1;
          while (fgets(in_line, inbuf) != 0)
              fputs(in_line, outbuf);
          fclose(inbuf);
        }
    /* now copy new message to inbox */
    if(fopen("b:tempin.dat", inbuf) == -1)
        { help(116), exit();}
    while(fgets(in_line, inbuf) != 0)
        fputs(in_line, outbuf);
    fclose(inbuf);
    fputs("\n^\n", outbuf);
    putc(26, outbuf);
    fflush(outbuf);
    fclose(outbuf);
    /* now close temp file and rename it */
    if (valid_inbox)
        unlink("b:inbox.dat");
    rename("b:tempbox.dat", "b:inbox.dat");
}
return(!eomsg);
}

```



```

/*****
/*
/*          LOCAL EMAIL SYSTEM
/*
/*          Author:  R. McDowell      (c)  1983.
/*
/*
*****/

```

```

/*****
/*
/*      This section contains the rest of the
/*      modules that are necessary to talk with
/*      the mainframe computers. The procedures
/*      contained here are:
/*      convert - convert msg from ext.
/*                  to internal format
/*      stfind - find a string in a given
/*                  buffer
/*      get_param - get a parameter from a
/*                  given string
/*      sendoutbox - send a file of
/*                  messages to external sys
/*      logout - logoff an external system
/*      findsys - find the system section
/*                  in commands file
/*      fsub - substitute a string for a
/*                  parameter token
/*      findchar - find a character within
/*                  a string
/*      get_sys - get the system within
/*                  the commands file
/*      get_cmd - get the command within
/*                  the commands file
/*
*****/

```

```

#include B:BDSCIO.H
#include b:variable.c

```

```

convert(to_system)
char *to_system;
{
char input_line[150], token_str[150], token_nums[80];
char string[150], parm[50], tempstr[150];
char tempbuf[BUFSIZ];
struct prms { char rparm[150]; } parameters[5];
int parm_no, ioerror, parms[6], index, length;
int position, sizmov;
fopen("b:command.dat", inbuf);
get_sys(to_system, inbuf);
get_cmd("analyse\n", inbuf);
fopen("b:tempout.dat", iniobuf);
strcpy(parameter[1], "");
strcpy(parameter[2], "");

```

```

strcpy(parameter[3], "\n");
strcpy(parameter[4], "");

fcreat("b:templ.dat", tempbuf);
while (1)
{
    /* read token number */
    fgets(input_line, inbuf);
    if (strcmp(input_line, "%\n") == 0)
        break;
    /* not end of header processing, get token to
       search for */
    parm_no = atoi(input_line);
    fgets(token_str, inbuf);
    /* remove the carriage return */
    length = strlen(token_str);
    token_str[length-1] = 0;
    /* now read lines until the token is found */
    while (1)
    {
        ioerror = fgets(input_line, iniobuf);
        if (ioerror == 0)
            {help(117); exit();}
        if ( (index = stfind(token_str, input_line))
            != 0)
        { /* have found token */
            /* remove token from the line */
            position = index + strlen(token_str) - 1;
            while (input_line[position] == ' ')
                position++;
            length = strlen(input_line);
            sizmov = length - position + 1;
            movmem(&input_line[position], input_line,
                    sizmov);
            /* now get information on all tokens
               needed */
            fgets(token_nums, inbuf);
            initw(parms, token_nums);
            /* now loop building up that
               parameter */
            parameter[parm_no].rparm[0] = 0;
            for (index = 1; index <= parms[0];
                index++)
            {
                get_parm(input_line, parms[index],
                           tempstr);
                strcat(parameter[parm_no].rparm,
                       tempstr);
                if (index != parms[0] )
                    strcat(parameter[parm_no].rparm,
                           "-");
            }
            strcat(parameter[parm_no].rparm, "\n");
            break;
        }
    }
}

```

```

    }
    else
    {
        /* not a known line, output if
           not blank */
        if (scan_string(input_line))
            fputs(input_line, tempbuf);
    }
}
putc(26, tempbuf);
fflush(tempbuf);
fclose(tempbuf);
fopen("b:templ.dat", tempbuf);

/* get the internal name for this sender */
int_name(parameter[2], parameter[2]);

/* now output the parameters and actual text */
fcreat("b:tempin.dat", outbuf);
for (index = 1; index <= 4; index++)
    fputs(parameter[index].rparm, outbuf);
while (1)
{
    if (fgets(input_line, tempbuf) != 0)
        fputs(input_line, outbuf);
    else
        break;
}
while (1)
{
    if (fgets(input_line, iniobuf) != 0)
        fputs(input_line, outbuf);
    else
        break;
}
putc(26, outbuf);
fflush(outbuf);
fclose(outbuf);
fclose(iniobuf);
fclose(tempbuf);
unlink("b:templ.dat");
}

int_name(external, internal)
char external[], internal[];
{
    char inbuf[BUFSIZ], inline[150], inputline[150];
    /* This routine will return a users internal name
       from the external one, open the user data file */
    if (fopen("b:users.dat", inbuf) == -1)
        exit();
    else

```

```

{
/* now loop through the file till the
   user is found */
while (fgets(inputline, inbuf) != 0)
{
    fgets(infile, inbuf);
    fgets(infile, inbuf);
    if (strcmp(infile, external) == 0)
    {
        /* have found name, return
           internal name */
        strcpy(internal, inputline);
        fclose(inbuf);
        return;
    }
    else
    {
        /* skip to next user */
        while (1)
        {
            fgets(inputline, inbuf);
            if (strcmp(inputline, "\n") == 0)
                break;
        }
    }
}

/* user not found, return external name */
strcpy(internal, external);
fclose(inbuf);
return;
}

scan_string(strg)
char strg[];
{
/* this routine will determine if a line is
   blank or not */
int len, index;
len = strlen(strg);
index = 0;
while ( len > index )
{
    if (isspace(strg[index]))
        index++;
    else
        return(1);
}
return(0);
}

stfind(token_str, input_line)
char *token_str, *input_line;
{

```

```

int index, index2, index3, length;
/* find how many characters to check */
length = strlen(input_line) - strlen(token_str);
index = 1;
while (length > index)
{
    index2 = 0;
    index3 = index - 1;
    while(token_str[index2] != 0 &
           token_str[index2] == input_line[index3])
        {index2++; index3++;}
    /* check for found or failure */
    if (token_str[index2] == 0)
        return(index);
    index++;
}
return(0);
}

```

```

get_parm(input_line, parm_no, string)
char *input_line, *string;
int parm_no;
{
    int lindex, index;
    lindex = 0;
    index = 1;
    if (parm_no == 0)
    { /* return entire line */
        strcpy(string, input_line);
    }
    else
    {
        while (index < parm_no)
        { /* skip over non blanks */
            while (input_line[lindex] != ' ' &
                   input_line[lindex] != '\n')
                lindex++;
            /* now check for end of line or
               blank character */
            if (input_line[lindex] == '\n')
            { /* parameter does not exist, return
               null string */
                strcpy(string, "");
                return;
            }
            else
            { /* now skip all blank characters */
                while(input_line[lindex] == ' ')
                    lindex++;
                index++;
            }
        }
        /* should now be at desired parameter, copy */
        index = 0;
    }
}

```

```

while(input_line[lindex] != ' ' &
      input_line[lindex] != '\n')
    string[index++] = input_line[lindex++];
string[index] = 0;
}

sendoutbox()
{
char inbuf[BUFSIZ], iniobuf[BUFSIZ];
char touser[50], tosystem[50], inputline[120];
char pause_str[120];
int sending, pause_time, ioerror;
int waittime;
char *parm[10], fromuser[50], copies[50], subject[120];
char currsys[30];
int sendout;

waittime = 4;
/* This routine will send messages from
   the outbox to the appropriate mainframe */

/* open the outbox */
if (fopen("b:outbox.dat", inbuf) == -1)
    return;

change_outbox(inbuf);
fclose(inbuf);
fopen("b:tempout.dat", inbuf);
sendout = 1;
currsys[0] = 0;

do
{
/* outbox exists, send message */
ioerror = fgets(touser, inbuf);
if (ioerror == 0)
    break;
else
fgets(fromuser, inbuf);
parm[2] = fromuser;
fgets(copies, inbuf);
fgets(subject, inbuf);
parm[3] = copies;
parm[4] = subject;

/* now have to find the system for
   this user */
findsys(touser, tosystem, parm[1]);

/* if system different, logoff
   current one. */
if (strcmp(tosystem, currsys) != 0)
{

```

```

        logout(currsys);
        movmem(tosystem, currsys,
            strlen(tosystem)+1);
        /* now logon to the system */
        login(tosystem);
    }

    /* now find the send command */
    fopen("b:command.dat", iniobuf);
    get_sys(tosystem, iniobuf);
    get_cmd("send\n", iniobuf);
    sending = 1;
    do
    {
        /* loop reading wait and
            send commands */
        fgets(pause_str, iniobuf);
        if(strcmp(pause_str, ""\n") == 0)
            sending = 0;
        else
        {
            pause_time = atoi(pause_str);
            fgets(inputline, iniobuf);
            fsab(inputline, parm);
            soutstr(inputline);
            pause_wait(pause_time);
        }
    }
    while(sending);

    /* now loop sending the text of
        the message */
    sending = 1;
    do
    {
        ioerror = fgets(inputline, inbuf);
        if ((ioerror==0) || (strcmp("", inputline) == 0))
        {
            sending = 0;
            if (ioerror == 0) sendout = 0;
        }
        else
        {
            soutstr(inputline);
            pause_wait(waittime);
        }
    }
    while(sending);

    /* now actually send the message on
        the mainframe */
    get_cmd("endsend\n", iniobuf);
    sending = 1;

```

```

do
{
/* loop reading commands */
fgets(pause_str, iniobuf);
if (strcmp(pause_str, ""\n") == 0)
    sending = 0;
else
{
    pause_time = atoi(pause_str);
    fgets(inputline, iniobuf);
    soutstr(inputline);
    pause_wait(pause_time);
}
while(sending);
fclose(iniobuf);
}
while (sendout);
fclose(inbuf);

/* now logoff the system */
logout(currsys);
unlink("b:tempout.dat");
}

change_outbox(inbuf)
char inbuf[];
{
char outbuf[BUFSIZ], out2buf[BUFSIZ];
char touser[50], tosystem[50], inputline[120];
char fromuser[50], copies[50];
char subject[90];
int ioerror, copynum;

fcreat("b:tempout.dat", outbuf);
while (1)
{
/* Transform all messages in the file */
ioerror = fgets(touser, inbuf);
if (ioerror == 0)
    break;
fgets(fromuser, inbuf);
fgets(copies, inbuf);
fgets(subject, inbuf);
/* now save original message */
fcreat("b:tempmsg.dat", out2buf);
while (1)
{
    fgets(inputline, inbuf);
    fputs(inputline, out2buf);
    if (strcmp(inputline, ""\n") == 0)
        break;
}
putc(26, out2buf);

```



```

fflush(out2buf);
fclose(out2buf);

/* now create first message */
fputs(touser, outbuf);
fputs(fromuser, outbuf);
fputs("\n", outbuf);
fputs(subject, outbuf);
fputs("copies to: ", outbuf);
fputs(copies, outbuf);
fputs(" \n", outbuf);
fopen("b:tempmsg.dat", out2buf);
while (fgets(inputline, out2buf) != 0)
    fputs(inputline, outbuf);
fclose(out2buf);

/* now create the copies for the users */
copynum = 1;
while (1)
{
    gettoken(copies, copynum, inputline);
    if (strcmp(inputline, "") == 0)
    { /* end of copies, terminate loop */
        break;
    }
    else
    {
        copynum++;
        /* create message for this user */
        fputs(inputline, outbuf);
        fputs("\n", outbuf);
        fputs(fromuser, outbuf);
        fputs("\n", outbuf);
        fputs(subject, outbuf);
        fputs("original to: ", outbuf);
        fputs(touser, outbuf);
        fputs("copies to: ", outbuf);
        fputs(copies, outbuf);
        fputs(" \n", outbuf);
        fopen("b:tempmsg.dat", out2buf);
        while (fgets(inputline, out2buf) != 0)
            fputs(inputline, outbuf);
        fclose(out2buf);
    }
}

/* outbox is now transformed, send it */
putc(26, outbuf);
fflush(outbuf);
fclose(outbuf);
}

gettoken(input_line, parm_no, string)
char *input_line, *string;

```

```

int parm_no;
{
int lindex, index;
lindex = 0;
index = 1;
if (parm_no == 0)
{ /* return entire line */
strcpy(string, input_line);
}
else
{
while (index < parm_no)
{ /* skip over non blanks */
while (input_line[lindex] != ',' &
input_line[lindex] != '\n')
lindex++;
/* now check for end of line or
blank character */
if (input_line[lindex] == '\n')
{ /* parameter does not exist,
return null string */
strcpy(string, "");
return;
}
else
{ /* now skip all blank characters */
lindex++;
while(input_line[lindex] == ' ')
lindex++;
index++;
}
}
/* should now be at desired parameter, copy */
index = 0;
while(input_line[lindex] != ',' &
input_line[lindex] != '\n')
string[index++] = input_line[lindex++];
string[index] = 0;
}
}

logout(currsys)
char currsys[];
{
/* This routine will log the user off a
given system. */
char iniobuf[BUFSIZ];
char inputline[150], pause_str[15];
int logging, pause_time, ioreturn;
if (strlen(currsys) != 0)
{
/* open the command file */
fopen("b:command.dat", iniobuf);

```

```

get_sys(currsys, iniobuf);
get_cmd("logout\n", iniobuf);

/* now send commands to logoff the system */
logging = 1;
do
{
    /* loop sending logout commands */
    fgets(pause_str, iniobuf);
    if (strcmp(pause_str, "\n") == 0)
        logging = 0;
    else
    {
        pause_time = atoi(pause_str);
        fgets(inputline, iniobuf);
        soutstr(inputline);
        pause_wait(pause_time);
    }
}
while(logging);
/* now close the files */
fclose(iniobuf);
drop_line();
}

findsys(touser, tosystem, userid)
char touser[], *tosystem, *userid;
{
    char inbuf[BUFSIZ], inputline[120];
    /* this routine will locate the system and
       userid for a person */

    /* open user data file */
    if (fopen("b:users.dat", inbuf) == -1)
    { help(118); exit(); }
    else
    {
        /* now loop through the file till user
           is found */
        while(fgets(inputline, inbuf) != 0)
        {
            if (strcmp(inputline, touser) == 0)
            { /* have found user, return system
               name and id */
                fgets(tosystem, inbuf);
                fgets(userid, inbuf);
                fclose(inbuf);
                return;
            }
            else
            { /* skip to next user */
                while(1)
                {
                    fgets(inputline, inbuf);

```

```

        if (strcmp(inputline, "^\n") == 0)
            break;
    }
    fclose(inbuf);
}

fsub(inputline, parm)
char inputline[], *parm[];
{
    int searching, index, parmno, startpos, num_move;
    char result[119];
    /* check for location of parameters */
    searching = 1;
    while(searching)
    {
        index = findchar('&', inputline);
        if (index == 0)
            searching = 0;
        else
        {
            /* have to replace parameter */
            parmno = atoi(&inputline[index]);
            /* now move lines around */
            /* first start of string */
            movmem(inputline, result, index-1);

            /* now the parameters */
            movmem(parm[parmno], &result[index-1],
                    strlen(parm[parmno])-1);
            startpos = strlen(parm[parmno]) + index - 2;
            /* now the remainder of the string */
            num_move = strlen(inputline) - index;
            movmem(&inputline[index+1], &result[startpos],
                    num_move);

            /* now put all back as inputline */
            movmem(result, inputline, strlen(result)+1);
        }
    }
}

findchar(inchar, string)
char inchar, string[];
{
    int i;
    i = 0;
    while (string[i] != inchar)
        if(string[i++] == '\0')
            return(0);
    return(i+1);
}

```

```

get_sys(system_name, iobuffer)
char system_name[], iobuffer[];
{
    int looking, ioerror;
    char inputline[150];

    /* This routine will advance the systems command
       file to the instructions for a given system. */
    looking = 1;
    do
    {
        ioerror = fgets(inputline, iobuffer);
        if (ioerror == 0)
            { help(119); return(-1); }
        else
        {
            if (strcmp(system_name, inputline) == 0)
                return(0);
            else
            {
                /* Not right system, skip to next */
                do
                {
                    ioerror = fgets(inputline,
                                     iobuffer);
                    if (ioerror == 0)
                        { help(120); return(-1); }
                }
                while (strcmp(inputline, "\n") != 0);
            }
        }
    } while (looking);
}

get_cmd(cmd_name, iobuffer)
char cmd_name[], iobuffer[];
{
    int looking, ioerror;
    char inputline[150];

    /* This routine will advance the commands file to
       the particular command desired. This assumes that
       get_sys has already found the desired system. */
    looking = 1;
    do
    {
        ioerror = fgets(inputline, iobuffer);
        if (ioerror == -1)
            { help(121); return(-1); }
        else
        {
            if (strcmp(cmd_name, inputline) == 0)
                return(0);
        }
    }

```

```
else
{
do
{
ioerror = fgets(inputline,
iobuffer);
if(ioerror == 0)
return(-1);
}
while (strcmp("", inputline) != 0);
}
}
while (looking);
}
```

Appendix F

Sample Run

A>b:email

E M A I L O F F I C E S Y S T E M

(R. McDowell (c) 1983.)

Password: russ

Command: l

Message number: 1

To: Russ

From: Prof. Kaye

cc:

Subject: Sample Data Run

Command: r

Message number: 1

To: Russ

From: Prof. Kaye

Sample Data Runs

Is it possible to include a sample data run of the
electronic messaging system?

Action: a

A>EDIT B:TEMPIN.DAT

-- at this point the user is in the local editor creating his
response to the message. On exiting the editor, the
session continues --

A>B:EMAIL RUSS 1

E M A I L O F F I C E S Y S T E M

(R. McDowell (c) 1983.)

Command: n

Box name: b:outbox.dat

Command: l

Message number: 1

To: Dan

From: Russ

cc:
Subject: Test of the local mail system

Message number: 2
To: Prof. Kaye
From: Russ
cc:
Subject: Re: Sample Data Runs

Command: e

System name: cocos

Please be patient, this will take a couple of minutes.

Command: u
Russ
cocos
R. McDowell
Account for cocos use

Prof. Kaye
envoy
ROGERK
Chair Of Office Automation
Carleton University

Command: x

Please be patient, this will take a couple of minutes.

A>

END

1 2 0 6 8 6

FIN