May 1983

PMDF - A PASCAL-Based Memo Distribution Facility

Ira Winston
University of Pennsylvania, IRA@CIS.UPENN.EDU

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-83-11.

PMDF - A PASCAL-Based Memo Distribution Facility

Abstract

This paper describes the implementation of PMDF, a portable Pascal-based internetwork mail router. It includes features such as aliasing, forwarding, queueing automatic routing to network gateways, message batching and message retransmission and is currently being used as a gateway between the Computer Science Network's PhoneNet and local electronic mail systems.

PMDF is unique among internetwork mailers in that it is portable. All operating system dependent functions have been isolated to one module and therefore PMDF can be ported to any system that has a Pascal compiler and provides the functions necessary to implement the operating system dependent module.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-83-11.

PMDF-A PASCAL-BASED MEMO DISTRIBUTION FACILITY

Ira Winston
MS-CIS-83-11

Department of Computer and Information Science
Moore School/D2
University of Pennsylvania
Philadelphia, PA 19104

May 1983

MS-CIS-83-11

UNIVERSITY OF PENNSYLVANIA
THE MOORE SCHOOL OF ELECTRICAL ENGINEERING
SCHOOL OF ENGINEERING AND APPLIED SCIENCE

PMDF - A PASCAL-BASED MEMO DISTRIBUTION FACILITY

Ira Winston

Philadelphia, Pennsylvania

May 1983

A thesis presented to the Faculty of Engineering and Applied Science of the University of Pennsylvania in partial fulfillment of the requirements for the degree of Master of Science in Engineering for graduate work in Computer and Information Science.

_____
O. Peter Buneman, Thesis Advisor

_____
O. Peter Buneman, Graduate Group Chairperson

ABSTRACT

This paper describes the implementation of PMDF, a portable Pascal-based internetwork mail router. It includes features such as aliasing, forwarding, queueing automatic routing to network gateways, message batching and message retransmission and is currently being used as a gateway between the Computer Science Network's PhoneNet and local electronic mail systems.

PMDF is unique among internetwork mailers in that it is portable. All operating system dependent functions have been isolated to one module and therefore PMDF can be ported to any system that has a Pascal compiler and provides the functions necessary to implement the operating system dependent module.

ACKNOWLEDGEMENTS

I would like to thank Sharon Perl, Mark Reinhold, Aravind Joshi and my advisor, Peter Buneman, for their valuable input to this project. I would also like to thank Dave Crocker for MMDF, the inspiration for PMDF, G. Brendan Reilly, Dave Farber and Eric Allman for their assistance with the Unix version and Michael Huhns, Steve Gutfreund and Sten Andler for testing early versions of PMDF.

Special thanks to the users of the CIS VAX-11/780 for putting up with PMDF even when it lost their mail, to Ruzena Bajcsy for her gentle persuations and, most of all, to Flaura for just about everything.

Table of Contents

1.0 INTRODUCTION

PMDF is a portable internetwork mail router, written in standard Pascal. Its primary use is as a gateway between the Computer Science Network's (CSNet) PhoneNet, a telephone-based electronic mail relay system, and local electronic mail systems. Features include aliasing and forwarding, queueing, automatic routing to network gateways, message batching and message retransmission.

PMDF implements a subset of the functions provided by the Multi-channel Memo Distribution Facility (MMDF) [Crocker79], developed by the Department of Electrical Engineering at the University of Delaware and used by the CSNet PhoneNet relay systems. MMDF is written in "C" and is heavily dependent upon the Unix* environment. PMDF was developed for VAX/VMS**, but because it is written in standard Pascal and because the operating system specific functions have been isolated to one module, it has been successfully ported to several other systems with little or no difficulty.

---

* Unix is a trademark of Western Electric

** VAX/VMS is a trademark of Digital Equipment Corporation

2.0 INTERNETWORK MAIL ROUTERS

An internetwork mail router provides a centralized "post office" to which all mail can be submitted and then subsequently be delivered, possibly to different networks, using the correct mailers for the specified destinations. It also performs message header and address rewriting as required.

Internetwork mail routers attempt to cope with the different message and address formats used by different networks by translating them from one format to another as the messages are moved between networks. Internet standards [Crocker82b] attempt to solve part of this problem by providing universal message and address formats.

The internet standards use a hierarchical addressing scheme known as "domain-based addressing" [Su82]. The hierarchy is a directed graph with a single path from the root of the tree to any node in the hierarchy. The root node is common to all addresses and is never referenced. Its children are "top-level" name-domains, which are the names of all the networks in the addressing universe. These network names are universally known, but domain names below the network name level are known only within the corresponding network. This serves to limit the number of names a host must know, but still allows any host in any

network to address any other host.

For example, in the internet standard address "ira@upenn.udel-relay.arpa", "arpa" is a network name, "udel-relay" is a host known in the "arpa" domain and "upenn" is a host known in the "udel-relay" sub-domain of the "arpa" domain.

Internetwork mail routers are needed even if standard internet addressing is used. Different networks still communicate using different mechanisms (communications equipment, message transmission protocols, etc.) and the internetwork mail router is used to select the correct delivery mechanism for a given message.

2.1 Sendmail

Sendmail [Allman83a, Allman83b] is a general internetwork mail router that includes facilities such as aliasing and forwarding, address rewriting, automatic routing to network gateways, queueing, message batching and flexible configuration. Its major accomplishment is in providing routing and address rewriting services that allow networks that do not conform to internet standards to communicate with other networks. It can handle old-style arbitrary address syntaxes as well as the newer domain-based internet addressing.

Sendmail does not interact with the user and does not perform actual mail delivery. It simply passes messages from one mailer to another, rewriting the header as necessary. One of Sendmail's weaknesses is its inability to provide "passive" delivery service. Sendmail must always initiate a delivery attempt, preventing it from routing mail through a communication channel established by a mailer. Sendmail is written in "C" and is heavily dependent on the Unix* environment.

2.2 MMDF

MMDF [Crocker79] is an internetwork mail router that includes facilities such as aliasing and forwarding, automatic forwarding to gateways, queueing, message batching, message timeout and a PhoneNet mailer. Its major function is providing an integrated view of a communication environment which can have a variety of different transmission channels. MMDF does not deal with the problem of address rewriting for a heterogeneous set of networks not conforming to the internet standards. It can only deal with network addresses that conform to a specific standard; currently the old Arpanet standard, RFC 733, is used although this will change to RFC 822, the new internet standard, soon. MMDF only handles heterogeneous networks that differ in type of communications equipment and message

transmission protocols. It does not handle heterogeneous message formats as does Sendmail.

The major difference between MMDF and Sendmail is the method used to determine which mailer (in Sendmail) or channel (in MMDF) to use in delivering a message. MMDF bases this decision on the host name, which is the string after the right most "@" in an address for RFC 733. It then locates the host name in the host tables it maintains for each channel in order to decide which channel to use. Sendmail, instead, uses a set of configurable production system rewriting rules to select the correct mailer and to rewrite the address. This allows Sendmail to handle a variety of address syntaxes.

2.3 PMDF

PMDF is a subset of MMDF and therefore it, too, can only handle heterogeneous networks that differ in type of communications equipment and message transmission protocols. It cannot handle heterogeneous message formats. PMDF has the potential to be a general internetwork mail router although it currently only provides two types of delivery channels; CSNet PhoneNet and local. PMDF uses MMDF's strategy of determining the delivery channel by looking up the host name in a host table. It provides facilities such

as aliasing and forwarding, queueing and message batching and provides only minimal message header rewriting services such as adding fields which note that it has routed a message.

PMDF's primary advantage over Sendmail and MMDF is its portability. Given its portability constraints, PMDF can never be expected to be the bridge between old style (ad hoc) and new style (internet standard) address syntaxes that Sendmail is. However, it will be able to handle addresses that conform to the new internet standards.

3.0 PHONENET

CSNet uses a variety of existing communication networks to provide high level network services to computer science research groups. Phone company lines are used directly for the PhoneNet service (a telephone based mail relay system) and indirectly through the value-added services of Telenet or Arpanet. CSNet provides additional protocols above the physical connections or packet-level services provided by the existing communication services [Landweber81]. Currently, the only service that CSNet provides is electronic mail transmission.

Most of the sites that are members of CSNet only have PhoneNet service. These sites exchange messages with each other and with Arpanet/Telenet sites via a PhoneNet relay machine. The PhoneNet relay machine calls the PhoneNet-only site on a regular basis to pick up and deliver messages. Alternatively, if the PhoneNet-only site has auto-dialing equipment, it can call the PhoneNet relay and exchange messages itself [Landweber81].

Two levels of protocol are used to implement the PhoneNet service. The lower level ("link-level") allows for transmission of arbitrary 7-bit ASCII data over ordinary telephone connections in a half or full duplex environment. The higher-level protocol is responsible for the details of

mail transfer. It must establish the context, and then pass individual messages [Crocker82a].

3.1 Link-level Protocol

The "link-level" provides services for an "active" or "master" host to place a phone call using auto-dialing equipment to a "passive" or "slave" host. The "master" host is able to login to the "slave" host as if it were a regular timesharing user and then issue the appropriate command to initiate the protocol program. This approach allows existing dial-in ports to be used for mail service. The login and protocol startup sequence is driven by a "script" file that resides on the "master" host [Szurkowski80].

Information is exchanged between "master" and "slave" in unit of packets. A packet contains a checksum, a type, a sequence number, flags and possibly type specific data. Packets are terminated with a carriage return/line feed combination. Every packet that is correctly received is acknowledged by the recipient. Logical packets, which are composed of one or more physical packets, can be used to transfer information that won't fit into one physical packet. The last physical packet in a logical packet will have the "end of segment" flag set.

The "link-level" protocol provides a mechanism for translating 7-bit ASCII characters that cannot be transmitted, into a sequence of legal characters, beginning with a predetermined escape character followed by two hexadecimal digits that are the ASCII value of the character to be transmitted.

Before any messages are exchanged, the "master" and "slave" exchange information about legal character sets (both input and output), maximum physical packet length and escape characters. The sequence of events is as follows:

1. The "slave" sends an XPATH packet giving the maximum packet length it can transmit and the characters it cannot send.

2. The "master" acknowledges with an XPATHACK packet.

3. The "slave" sends an RPATH packet with the maximum packet length it can receive and the characters that it cannot receive.

4. The "master" acknowledges with an RPATHACK.

5. The "master" sends its XPATH packet.

6. The "slave" acknowledges with an XPATHACK.

7. The "master" send its RPATH packet.

8. The "slave" acknowledges with an RPATHACK.

9. The "slave" sends an ESCAPE packet.

10. The "master" acknowledges with and ESCAPEACK.

11. The "master" sends an ESCAPE packet.

12. The "slave" acknowledges with and ESCAPEACK.

After the startup sequence is complete, messages are exchanged using DATA and DATAACK packets.

There is no negative acknowledgement packet type. Instead, if the sender of a packet does not receive an acknowledgement within a reasonable time after transmission, the original packet is retransmitted. Several retransmissions are attempted and if there is still no response, the protocol program is aborted. Acknowledgements have the same sequence number as the packet they are acknowledging. For a complete description of the "link-level" protocol see [Szurkowski80].

3.2 Message Protocol

In the higher level protocol, the "master" has three commands available: "submit", to send mail to the "slave"; "pickup", to pick up mail from the "slave"; and "end", to terminate the session. When the "master" sends a command, the slave returns an MMDF reply code. All information, including actual message text, is sent in individual logical packets.

An individual message is processed as follows. The first logical packet of message contains the return address. The next set of packets contains, the addresses of the recipients, one per packet. An MMDF reply code is returned after each address is sent to indicate whether or not the address was accepted. After all of the addresses have been sent, a null packet is sent indicating that the address list is complete. The message text is then sent as one logical packet consisting of one or more physical packets. After the message is received a reply packet is returned indicating the final status of the mail transfer.

The next message may be sent by repeating the cycle; the submitter will send a null packet when there are no more messages to be sent.

A reply value consists of an 8-bit MMDF reply code encoded as two hexadecimal digits and string describing the error. The reply codes are listed in Appendix B.

4.0 DESCRIPTION OF PMDF

The functions of PMDF can be separated into two parts, message submission and message delivery. The message submission section provides a centralized "post office" for submission of all messages. The message delivery section selects the correct delivery channel to deliver a message and then performs the delivery using that channel.

Each delivery channel consists of a message queue which holds messages for the channel and a program that delivers messages from the queue to their destination (or to the next stop in the path to their destination). A local channel delivers mail to a local electronic mail system; Pobox channels deliver mail to PhoneNet sites that dial-in to pickup mail; and Phone channels dial-out to PhoneNet sites to deliver mail. The Pobox and Phone channel programs also submit mail from remote PhoneNet hosts to the local system.

PMDF is composed of several channel programs, user interface programs and subroutine packages. The mm_ subroutine package is used for message submission and address parsing, the qu_ package is used for reading messages from message queues, the di_ package implements the PhoneNet "link level" protocol, the ut_ package provides string and file handling functions and the os_ package contains the operating system dependent routines.

The SEND program is a user interface program that allows users to compose messages to be submitted to PMDF. It uses the mm_, ut_ and os_ packages.

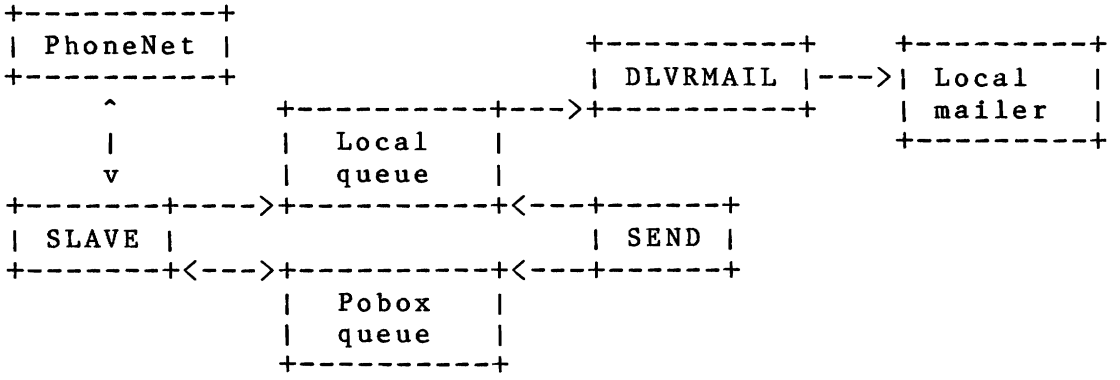
The SLAVE program is the channel program for Pobox channels and is invoked when a PhoneNet site dials in to the local system to pickup and deliver mail. It uses the mm_, di_, qu_, ut_ and os_ packages.

The MASTER program is the channel program for Phone channels and is used to dial-out to a remote PhoneNet site to deliver and pickup mail. It uses the mm_, di_, qu_, ut_ and os_ packages.

The DLVRMAIL program is the channel program for the local channel. It uses the qu_, ut_ and os_ packages.

4.1 Mail Flow

The following diagram illustrates the flow of mail through a hypothetical PMDF configuration with one local channel and one Pobox channel:



4.2 mm_ Package

4.2.1 Channel Configuration -

One of the inputs to the mm_ package is a channel configuration file which describes the channels present on the system. There is one entry in the file for each defined channel. Each entry includes a one-letter channel code, an official host name, a list of synonyms for the official name and a list of the names of hosts for whom the channel acts as a relay. This information is used by the mm_ package to determine to which channel a particular address refers. The exact format of the channel configuration file is described in Appendix G.

4.2.2 Aliasing -

Aliasing is a facility that allows local addresses to be automatically translated to one or more different local or network addresses. The alias file defines these translations and consists of a series of entries of the form "alias: address,[...]" (for example, "info-pmdf: jsmith,jdoe"). When a local address is processed that matches one of the aliases in the alias file, the address is replaced by the contents of its corresponding alias entry. A line of the form "alias: <filename" will use the contents of 'filename' for the list of alias values. Alias file entries can reference other aliases, but only forward references are allowed.

Aliases may be used for one of three purposes:

1. True aliasing, where the alias value is a local user's login name, e.g. "john: jsmith".

2. Forwarding, where the alias value is a foreign address, such as "jsmith: jsmith@UDel".

3. Address lists, where the alias value is a set of addresses, such as "info-pmdf: jsmith,jdoe".

4.2.3 The Message Submission Process -

All messages are submitted through the mm_ package in three parts: first the return address; then a list of recipients; and finally the message text. Before any messages are submitted mm_init must be called to initialize the package. It reads the alias file as well as the channel configuration file.

The first step of the message submission process is to call mm_winit (initialize for writing) with the return address and source channel name. mm_winit records this information for later use. Then, one address at a time is passed to mm_wadr (write one address). Each address is analyzed according the RFC 733 [Crocker77] address standards.

The string after the rightmost "@" in an address is called the host name, which is used to determine to which channel the address refers. If the host name refers to the local channel, then the alias file is consulted to see if the address is an alias. If it is an alias, then the address is replaced by its alias value. Otherwise, mm_wadr determines whether the address refers to the login name of a local user.

If the host name does not refer to the local channel, then the host name tables for the other channels are consulted and the channel to which this host name refers is determined.

If the address passed to mm_wadr is valid, it is added to a list of addresses for the channel to which it refers. If the address is invalid, an appropriate error code is returned. An alias is considered valid if any of its translations are valid.

After all of the addresses have been passed to mm_wadr, mm_waend (write address end) is called.

Next, the text of the message is passed to mm_wtxt (write text), one line at a time. mm_wtxt copies the text to a temporary file. The text of a message consists of header lines and an optional message body. Header lines are of the form "field: value" and may be continued by starting the continuation line with one or more blanks. The message body is a sequence of lines containing ASCII characters. The two are separated by a null line.

The only special processing performed by mm_wtxt is the insertion of a "Received:" header line at the end of the message header. The "Received:" header line indicates from which host the message was received and to which host it was

submitted. This is only done for messages that are submitted from a remote system (determined from the source channel name that was passed to mm_winit).

After all the text has been processed by mm_wtxt, mm_wtend (write text end) is called to create message files in the queues of all channels that have non-empty address lists.

A message file consist of five sections:

1. the return address (saved by mm_winit),
2. a list of recipient addresses for the channel (accumulated by mm_wadr);
3. an end-of-address sentinel;
4. the message body (read from the temporary file created by mm_wtxt); and
5. an end-of-text sentinel.

Only those recipient addresses that refer to a specific channel are written to that channel's message file.

The end-of-text sentinel protects against delivery of partially written message files created by failures because incomplete messages (without end-of-text sentinels) are not delivered. The exact format of a message file format is

described in Appendix H.

The mm_ package also includes an RFC 733 [Crocker77] address parser which is used by mail composition programs such as the SEND program.

Appendix C contains a complete specification of the mm_ routines and Appendix D illustrates sample message submission code.

4.3 qu_ Package

The qu_ package provides a standard mechanism for channel programs to retrieve messages from their queues. There is no facility in the qu_ package by which a channel can obtain a list of the filenames of the messages in its queue because this primitive would be hard to provide in a portable fashion. Instead, a channel program expects, as input, a file listing the filenames of the messages. If possible, this list of filenames should be sorted by creation time, so that messages are delivered in the same order that they were submitted. This places the burden of producing the queue listing on an external program or command procedure.

The qu_ package is initialized by a call to qu_init. A message file is opened by calling qu_rinit (initialize for reading) with a filename and the name of the channel. If qu_rinit can open the specified message file then it returns the return address which is read from the message file otherwise, it returns an appropriate error code.

The recipient addresses are then read, one at a time, using qu_radr (read one address) which returns an "end of file" status when all of the addresses have been read.

qu_rtxt (read text) reads the message text and returns text or an "end of file" indication.

The entire process is terminated by calling qu_rend (end reading) to delete the message file which has presumably just been delivered. qu_rkill (kill reading) can be used to terminate message reading without deleting the message file.

Appendix E contains a complete specification of the qu_ routines.

4.4 di_ Package

The di_ package is an implementation of the PhoneNet "link level" protocol. di_init sets up the dial package. di_snd_rpath, di_snd_xpath, di_rcv_rpath, di_rcv_xpath,

di_snd_escape and di_rcv_escape are used to exchange information about legal character sets, (both input and output) maximum packet length, and escape characters.

Characters that cannot be received by the remote system and those that cannot be transmitted by the local system are replaced by the escape character followed by two hexadecimal digits (the ASCII representation of the character to be transmitted) by di_packet_encode. The packet can then be transmitted by di_packet_write, which waits for the packet to be acknowledged. If the packet is not acknowledged in a reasonable amount of time, di_packet_write retransmits the packet and waits for an acknowledgement. Several retries are attempted.

di_read_packet reads a packet from the remote system. If a packet is not received within a reasonable period of time, di_read_packet aborts the program. di_read_packet also checks packets for correct format, type, sequence number and checksum and acknowledges correctly received packets. di_packet_convert can then be used to translate escaped characters in a packet to the corresponding untranslated characters.

di_read_record reads a logical record consisting of one or more physical packets, the last of which has the "end of segment" flag set, from the remote system.

The di_ package produces a transcript file with a record of all packets received and sent and an error log with a record of all timeouts and incorrectly received packets.

Appendix F contains a complete specification of the di_ routines.

4.5 os_ Package

All operating system dependent functions are isolated in the os_ package to simplify the process of porting PMDF to different systems. The following services are included:

1. Supplying the current date, time and time zone, (needed for logging and message header composition);

2. Pausing for a specified number of seconds, (used by the MASTER program in script processing);

3. Determining the validity of a specified login name, (used by the mm_ package for address validation for the local channel and by the DLVRMAIL program for

delivering local messages);

4. Reading a line, terminated by a carriage return/line feed pair, from a terminal and returning an error status if the line is not received within a specified number of seconds, (used by the di_ package);

5. Reading one character from a terminal and returning an error status if the character is not received within a specified number of seconds, (used by the MASTER program during script processing);

6. Writing a string to a terminal without adding any formatting characters, such as carriage returns and line feeds, or delay characters, such as nulls or rubouts, (used by the di_ package and the MASTER program during script processing);

7. Opening a file for reading or writing and returning an error code if the file cannot be opened, (used by all of the packages and programs); and

8. Creating a unique filename given a channel name, (used by the mm_ package to create unique filenames for the message queues).

4.6 The SLAVE Program

The SLAVE program is invoked when a remote PhoneNet host dials into the local system. One of the inputs to the program is a file containing a list of message filenames from the Pobox channel queue that are to be delivered to the remote PhoneNet site.

The SLAVE program uses the di_ package to exchange configuration information with the remote host. Messages are then picked up from the remote host using the MMDF message-level protocol described in chapter 2. These messages are submitted to PMDF through the mm_ package. Any rejected addresses are passed back to the remote host so that the message sender can be notified.

After the pick-up process is completed, messages are taken from the Pobox channel queue and are delivered to the remote host. If a message cannot be delivered by the remote host, a return message is sent to the sender of the message. This return message is composed by the SLAVE and submitted using the mm_ package.

The SLAVE program maintains a log file that contains information about received and delivered messages; size, sender, recipient and address rejections are recorded.

4.7 The MASTER Program

The MASTER program is very similar to the SLAVE program except that it is responsible for initiating the communications connection. This connection is established with the aid of a "script" file that specifies the sequence of actions required to establish a physical connection, to login to the remote host and to start up the SLAVE program on the remote host. A "script" file consists of a series of commands that tell the MASTER program what to send to the remote host and what responses to expect in return.

There are four basic commands; "xmit", "recv", "go" and "end". Each command appears on a separate line in the file with its arguments. The "xmit" command has one argument; a string of characters that will be transmitted to the remote host. The "recv" command has two arguments; a string to receive from the remote host and a maximum number of seconds to wait for a response from the remote host. The "go" command starts the message transfer section of the MASTER program and the "end" command terminates the program. "xmit" and "recv" commands appearing between the "go" and "end" commands are used to logout from the remote system in an orderly fashion before the phone connection is terminated.

Many popular auto-dialers expect dialing instructions as sequences of ASCII characters (including the phone number to be dialed) and also output sequences of ASCII characters which report the results of attempted calls. This class of auto-dialers can be controlled using the "xmit" and "recv" commands of the "script" file.

4.8 The DLVRMAIL Program

The DLVRMAIL program is the channel program for the local channel which is responsible for delivering messages from the local channel's message queue to local users. It reads messages from the queue using the qu_ package, and, when necessary, reformats them for the local mail system. Usually, a local mailer is somehow called to perform the actual delivery. In a simple mail system, messages might just be appended to a local user's mail file.

4.9 The SEND Program

The SEND program is a direct translation of the MMDF SEND program from "C" to Pascal with MMDF calls replaced by calls to mm_ routines and with Unix system calls replaced by calls to appropriate os_ routines. It is an interactive program that allows a user to compose a message for submission. For a complete description of the SEND program,

see the MMDF documentation [Crocker82a].

5.0 CURRENT IMPLEMENTATIONS

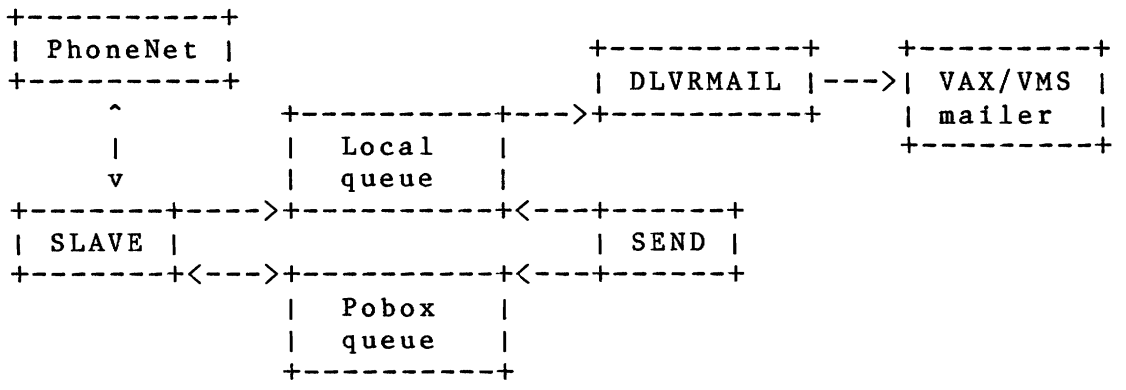
There are five CSNet Phonenet sites running the basic VAX/VMS version of PMDF using a Pobox channel for communication with a relay machine and a local channel to interface to the standard VAX/VMS mail program. Another VAX/VMS site runs an expanded version that also includes a dial-out Phone channel to a system running the basic version. PMDF also runs in a two-channel configuration on an IBM system under VM/CMS and on a Hewlett Packard 3000. A modified version of PMDF is used by Berkeley Unix* systems as a PhoneNet mailer that interfaces to Sendmail.

5.1 VAX/VMS Two-Channel Version

The standard VAX/VMS mail program cannot be used to submit messages to PMDF. It is used to deliver mail to local users but the PMDF SEND program must be used for submission. A message delivered to a local user appears as though it comes from a local user named "CSNET" with the subject field containing the name of the real sender, and the body of the message containing the other header lines as well as the body of the original message.

Before the SLAVE program exits, having just submitted and received messages, it starts up a detached process that runs the DLVRMAIL program to deliver any messages that have just arrived for the local channel. This is the only time that the DLVRMAIL program is run. It is assumed that all strictly local mail will be sent using the VAX/VMS mail program and therefore will not have to be delivered by DLVRMAIL.

The following diagram illustrates the flow of mail through the system:



5.2 VAX/VMS Three-Channel (Relay) Version

A VAX-11/780 in the Computer and Information Science Department at the University of Pennsylvania uses a three-channel system that communicates with a CSNet PhoneNet relay using a Pobox channel and with a local VAX-11/750

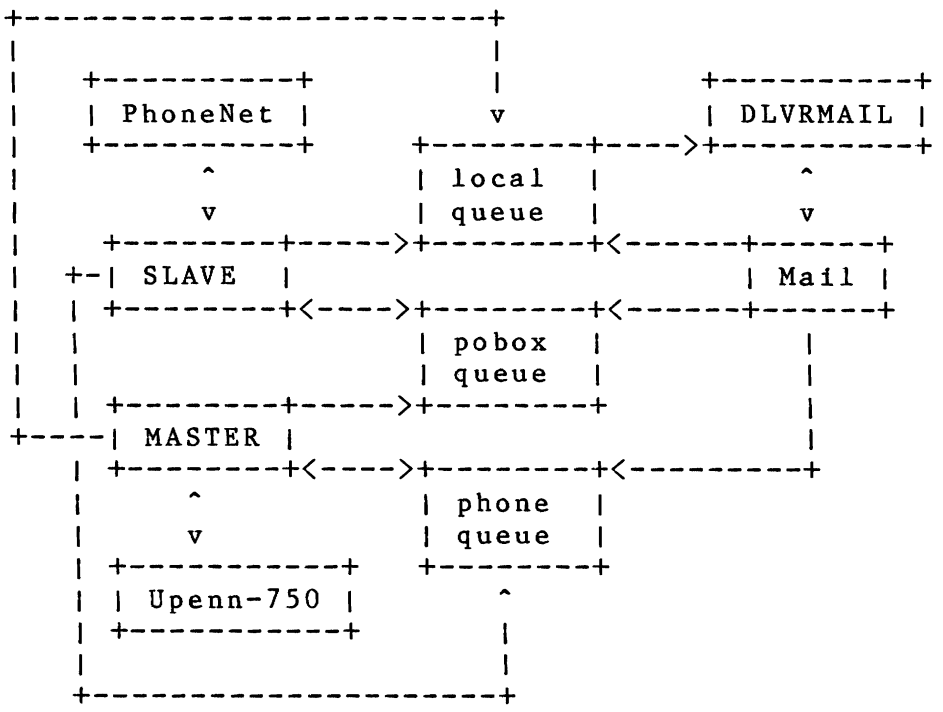
using a Phone channel. The VAX-11/780 acts as a relay for the VAX-11/750; that is, all network messages destined for the VAX-11/750 must first pass through the VAX-11/780.

The standard VAX/VMS mail program is not used on this system. Instead a more useable mail program has been developed [Perl82] that uses PMDF for all mail submission. All mail, local and network, is sent using the same mailer. The DLVRMAIL program has been modified to perform the local delivery required by the mail program. This involves appending a message to a user's central mail file, rewriting headers as necessary and notifying the recipient if he or she is logged in at the time of delivery.

Because PMDF is used for all local mail delivery in this configuration, the DLRVMAIL program must be run more frequently than in the two-channel configuration. This was accomplished by modifying the mm_ package to signal a detached process to run the DLVRMAIL program whenever a message is placed in the local channel's queue.

Mail is delivered to and received from the local VAX-11/750 by a detached process that runs the MASTER program every thirty minutes.

The following diagram illustrates the flow of mail through the system:



5.3 Berkeley Unix* Version

Currently a PhoneNet-only site that is running the Berkeley Unix* operating system uses MMDF to interface to the PhoneNet. The standard Unix* mail program does not provide an interface to MMDF and therefore cannot be used to submit messages directly to the PhoneNet.

Two solutions to this problem have been implemented; using the mail program provided with MMDF (xmsg) and using a modified version of Delivermail (Sendmail's predecessor) to

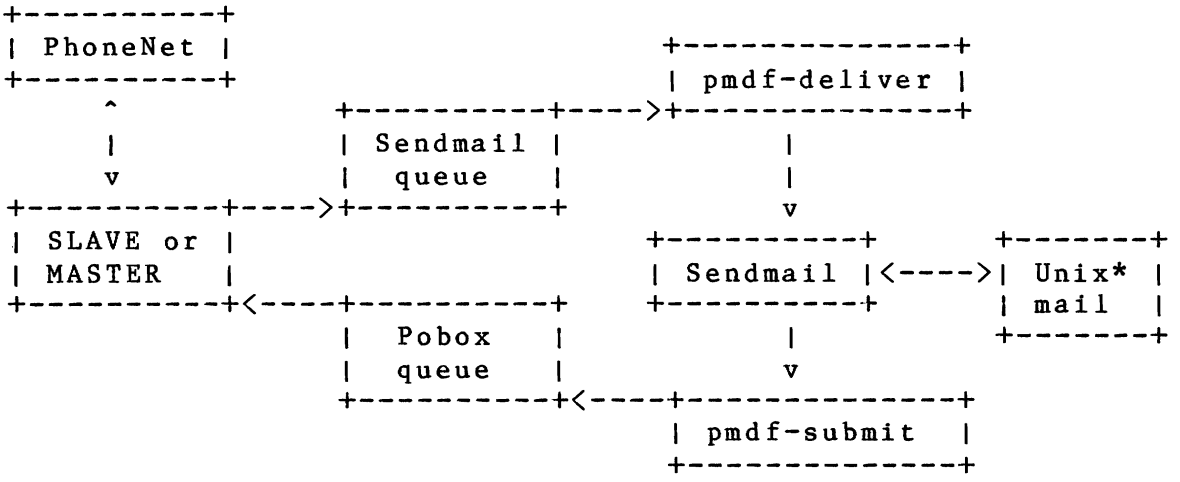
provide the link between Unix* mail and MMDF [Crocker82a]. The first solution is unacceptable in most environments and the second solution is considered to be too complicated and is extremely difficult to maintain. It should not be necessary to have two full-scale internetwork mail routers on the same system. The only feature of MMDF that is used is its PhoneNet mailer, yet the entire system must be installed, which is not desirable because of its size.

A new approach to the problem that uses PMDF as a PhoneNet mailer will be available in the near future. PMDF has been extended to act as a gateway between Sendmail and the PhoneNet. This variant of PMDF uses two channels; a Sendmail channel and a Pobox channel (or Phone channel if the MASTER program is used). The Pobox (or Phone) channel program is the SLAVE (or MASTER) program. It operates as usual, exchanging messages with the PhoneNet relay and submitting them to PMDF using the mm_ package.

The Sendmail channel program (pmdf-deliver) submits messages from its queue to Sendmail using the standard submission mechanism provided by Sendmail. Sendmail submits messages to PMDF using a mailer (pmdf-submit) that accepts messages and queues them using the mm_ package.

For this environment, address validation and alias file processing have been removed from the mm_ package because Sendmail already provides these functions. The channel selection logic has also been simplified so that all messages submitted from the PhoneNet channel are placed in the Sendmail queue and all messages submitted from the Sendmail channel are placed in the PhoneNet queue. Both of these simplifications are designed to allow Sendmail to perform the bulk of the processing and to only use PMDF as a PhoneNet mailer.

The following diagram illustrates the flow of mail through the system:



6.0 FUTURE DIRECTIONS

In addition to porting PMDF to several different systems, there are also plans to implement a number of enhancements.

6.1 Internet Standards (RFC 822)

The Internet standard message and address formats, as defined by RFC 822, will be used by Arpanet and CSNet hosts in the near future. When PhoneNet relay machines running MMDF start accepting RFC 822 style addresses from PhoneNet-only sites, it will be necessary to modify PMDF to generate messages that conform to RFC 822.

In preparation for this, an RFC 822 address parser, written in standard Pascal has been developed. It uses an SLR parser generator written at the University of Pennsylvania [Schimpf81]. The parser generator processes a BNF description of an input language (legal address forms, in this case) and produces an SLR parse table as output.

The mm_ package as well as the SEND and DLVRMAIL programs currently use an ad hoc parser to parse RFC 733 style addresses and will have to be modified to use the new address parser. In order to generate messages conforming to the RFC 822 message format, additional modifications will

have to be made to the mm_ package to include several new message header and address rewriting services.

The address parser is described in Appendix I.

6.2 DECNET

Digital Equipment Corporation's DECNET networking hardware and software allows a set of computers to communicate with one another at data rates ranging from 9600 baud to 1 megabaud [DEC82a]. If a set of machines that use DECNET were to join the CSNet PhoneNet, two configurations would be possible:

1. Each machine could exchange mail with an external relay independently.

2. One machine could act as a relay by exchanging mail with a PhoneNet relay and also exchanging mail with other machines attached to the local DECNET network.

If all sites were to use the first approach, then the PhoneNet relays would be severely overloaded. The second approach could be implemented by using Pobox and Phone channels with RS232 connections (phone or direct connect). A more intelligent approach would be to implement a DECNET

channel for PMDF using DECNET's task-to-task communication facility.

Instead of using the PhoneNet message level protocol to communicate between the nodes of a DECNET network, the new internet standard SMTP (simple mail transfer protocol) [Postel82] will probably be used.

7.0 REFERENCES

[Allman83a] Allman, E., "SENDMAIL - An Internetwork Mail Router", Unpublished Manuscript. March 1983.

[Allman83b] Allman, E., "Sendmail Installation and Operation Guide", Technical Report, Computer Science Department, University of California at Berkeley, March 1983.

[Crocker77] Crocker, D.H., Vittal, J.J., Pogran, K.T., and Henderson, D.A. Jr., "Standard for the Format of ARPA Network Text Messages". RFC 733, NIC 41952, In [Feinler78]. November 1977.

[Crocker79] Crocker, D.H., Szurkowski, E.S., and Farber, D.J., "An Internetwork Memo Distribution Capability -- MMDF", Proceedings of the Sixth Data Communications Symposium, Pacific Grove, CA. November 1979

[Crocker82a] Crocker, D.H., MMDF Documentation Package, Department of Electrical Engineering, University of Delaware, Newark, Delaware. March 1982.

[Crocker82b] Crocker, D.H., "Standard for the Format of Arpa Internet Text Messages". RFC 822. Network Information Center, SRI International, Menlo Park, California. August 1982.

[DEC82a] DECnet-VAX User's Guide, Digital Equipment Corporation, Maynard, Massachusetts, Massachusetts. May 1982.

[DEC82b] VAX-11 Pascal Language Reference Manual, Digital Equipment Corporation, Maynard, Massachussetts. October 1982.

[Feinler78] Feinler, E., and Postel, J. (eds.), ARPANET Protocol Handbook, NIC 7104, Network Information Center, SRI International, Menlo Park, California. 1978.

[Landweber81] Landweber, L.H., and Solomon, M., "Multiple Networks in CSNet", CSNet Design Note DN-1. November 1981.

[Perl82] Perl, S., "Mail User's Guide, Version 2.0", Technical Report MS-CIS-82-21, Computer and Information Science Department, University of Pennsylvania, Philadelphia, Pennsylvania. August 1982

[Postel82] Postel, J.B., "Simple Mail Transfer Protocol". RFC 821, Network Information Center, SRI International, Menlo Park, California. August 1982.

[Schimpf81] Schimpf, K.M., "Construction Methods of LR Parsers", Masters Thesis, Computer and Information Science Department, University of Pennsylvania, Philadelphia, Pennsylvania. May 1981.

[Su82] Su, Z., Postel, J., "The Domain Naming Convention for Internet User Applications". RFC 819. Network Information Center, SRI International, Menlo Park, California. August 1982.

[Szurkowski80] Szurkowski, Edward S., "MMDF Dial-up Link Protocol", CSNet Design note DN-4, April 1980.

APPENDIX A

DATA TYPES

The following Pascal type definitions are used in the descriptions of the PMDF subroutine packages:

```
alfa = PACKED ARRAY[1..ALFA_SIZE] OF char;

vstring = RECORD
  body: alfa;
  length: integer;
  END;

bigalfa = PACKED ARRAY [1..BIGALFA_SIZE] OF char;

bigvstring = RECORD
  body: bigalfa;
  length: integer;
  END;

vstringptr = ^vstring;

vstringlptr = ^vstringl;

vstringl = RECORD
  link: vstringlptr;
  val:  vstring;
  END;

short_alfa = PACKED ARRAY[1..SHORTALFA_SIZE] OF char;
```

```
shortvstring = RECORD
  length: integer;
  body: short_alfa;
  END;

rp_replyval = 0..255;          (* PMDF reply value *)

rp_bufstruct = RECORD          (* PMDF reply structure *)
  rp_val:  rp_replyval;
  rp_line: vstring;
  END;

di_bit = 0..1;

di_packet_type = 0..9;

mm_channel_ptr = ^mm_channel;

ch_chancode = char;

aliasptr = ^alias;
entryptr = ^entry;
alias = RECORD
  link: aliasptr;
  val:  shortvstring;
  member: entryptr;
  END;
entry = RECORD
  link:  entryptr;
  val: shortvstring;
  END;

hostptr = ^hostentry;
hostentry = RECORD;
  link:  hostptr;
  val: shortvstring;
  official: shortvstring;
  END;

mm_channel = RECORD
  link: mm_channel_ptr;
  address_list: vstringlptr;
  chancode: ch_chancode;
  host_table: hostptr;
  official_hostname: vstring;
  END;
```

APPENDIX B

MMDF STATUS CODES

```
RP_USER   =    216;    (* No such user      (D8) *)
RP_AOK    =     96;    (* Address ok        (60) *)
RP_MOK    =     32;    (* Message ok        (20) *)
RP_OK     =      9;    (* ok                (09) *)
RP_NDEL   =    224;    (* couldn't deliver  (E0) *)
RP_FCRT   =    169;    (* file create err   (A9) *)
RP_FOPN   =    170;    (* file open err     (AA) *)
RP_NO     =    201;    (* basic no          (C9) *)
RP_NOOP   =    138;    (* no operation      (8A) *)
RP_EOF    =    139;    (* end of file       (8B) *)
RP_AGN    =    136;    (* try again later   (88) *)
```

APPENDIX C

mm_ PACKAGE

Global variables -

mm_local_channel: mm_channel_ptr;

The channel configuration information.

mm_alias_list: aliasptr;

The alias information.

pmdf_config_file: text;

The file variable used to read the channel configuration file.

pmdf_alias_file: text;

The file variable used to read the alias file.

External functions -

All functions return an MMDF reply code as defined in Appendix 3.

```
FUNCTION mm_init: rp_replyval;
```

Initializes the mm_ package.

```
FUNCTION mm_sbinit: rp_replyval;
```

Initialize for submission.

```
FUNCTION mm_winit (vianet: char; retadr: vstring):
                  rp_replyval;
```

Initialize for the submission of a message. 'vianet' is the one letter channel code corresponding to the channel submitting the message. 'retadr' is the return address, that is, the address to which notification should be sent if the message cannot be delivered.

```
FUNCTION mm_wadr (host, adr: vstring): rp_replyval;
```

Write the address of one of the recipients of the message. 'host' and 'adr' specify the address. The return code from mm_wadr is only an indication of the accessibility of the message file. mm_rrply must be called after mm_wadr to determine if the address was accepted.

```
FUNCTION mm_waend: rp_replyval;
```

Terminate address list.

```
FUNCTION mm_wtxt (buf: vstring);
```

Write text to the message files associated with the selected channels.

```
FUNCTION mm_bigwtxt (buf: bigvstring): rp_replyval;
```

Write text from a bigvstring.

```
FUNCTION mm_wtend: rp_replyval;
```

Terminate text section. mm_rrply must be called after mm_wtend to determine the final status of the message submission.

```
FUNCTION mm_wkill: rp_replyval;
```

Prematurely terminate the submission of a message.

```
FUNCTION mm_rrply (reply: rp_bufstruct): rp_replyval;
```

Used to obtain status value and associated message after a call to mm_wadr or mm_wtend.

```
FUNCTION mm_end (typ: boolean): rp_replyval;
```

Terminate use of the mm_ package. If typ is true, the package is terminated normally. If it is false, all open message files are deleted and then the package is terminated.

APPENDIX D

mm_ SAMPLE SUBMISSION CODE

```
PROCEDURE send_manage: rp_replyval;

  BEGIN
    IF (rp_isbad(mm_init)) THEN abort_submit;
    IF (rp_isbad(mm_sbinit)) THEN abort_submit;
    WHILE more_messages DO
      IF rp_isbad(do_a_message) THEN abort_submit;
    send_manage := RP_OK;
    END;

PROCEDURE do_a_message: rp_replyval;

  BEGIN
    IF rp_isbad(mm_winit('l',sender)) THEN abort_submit;
    WHILE more_addresses DO
      IF rp_isbad(do_an_address) THEN abort_submit;
    IF rp_isbad(mm_waend) THEN abort_submit;
    do_a_message := do_text;
    END;

PROCEDURE do_an_address: rp_replyval;

  VAR thereply: rp_bufstruct;

  BEGIN
    IF rp_isbad(mm_wadr(host,adr)) THEN abort_submit;
    IF rp_isbad(mm_rrply(thereply)) THEN abort_submit;
    CASE thereply.rp_val OF
      RP_AOK: note_acceptance;
      RP_NO,
      RP_USER,
```

```
      RP_NDEL,
      RP_AGN,
      RP_NOOP: note_failure;
      OTHERWISE note_bad_replycode;
      END;
    do_an_address := RP_OK;
    END;

PROCEDURE do_text: rp_replyval;

  VAR thereply: rp_bufstruct;

  BEGIN
    WHILE more_text DO BEGIN
      get_text(buffer);
      IF rp_isbad(mm_wtxt(buffer)) THEN abort_submit;
      END;
    IF rp_isbad(mm_wtend) THEN abort_submit;
      IF rp_isbad(mm_rrply(thereply)) THEN abort_submit;
    CASE thereply.rp_val OF
      RP_MOK,
      RP_OK: note_acceptance;
      RP_NO,
      RP_USER,
      RP_NDEL,
      RP_AGN,
      RP_NOOP: note_failure;
      OTHERWISE note_bad_replycode;
      END;
    do_text := RP_OK;
    END;
```

APPENDIX E

qu_ PACKAGE

External functions -

All functions return an MMDF reply code as defined in Appendix B.

```
FUNCTION qu_init: rp_replyval;
```

Initialize the qu_ package.

```
FUNCTION qu_rinit (filename: vstring;
                   VAR return_address: vstring;
                   chancode: char): rp_replyval;
```

Initialize for the reading of a message file. 'filename' is the name of the message file to be processed. 'return_address' is set by qu_rinit to the return address which is read from the message file. 'chancode' is the one letter channel code of the channel whose message file is being read.

```
FUNCTION qu_radr (VAR addr: vstring):
                  rp_replyval;
```

Read the next address and return it in 'addr'. RP_EOF is returned when there are no more addresses.

```
FUNCTION qu_rtxt (VAR line: bigvstring):
                 rp_replyval;
```

Read the next line from the message file and return it in 'line'. RP_EOF is returned when there is no more text.

```
FUNCTION qu_rend: rp_replyval;
```

Terminate reading of a message and delete the corresponding message file.

```
FUNCTION qu_rkill: rp_replyval;
```

Terminate reading of a message but do not delete the corresponding message file.

```
FUNCTION qu_end: rp_replyval;
```

Terminate use of the qu_ package.

APPENDIX F

di_ PACKAGE

External Functions -

PROCEDURE di_init (progtype: di_bit);

Initialize the di package. 'progtype' should be 0 for the MASTER program and 1 for the SLAVE program and is used to set the origin bit in packets correctly.

PROCEDURE di_snd_xpath;

Send an XPATH packet to remote host and receive an XPATHACK packet in return.

PROCEDURE di_snd_rpath;

Send an RPATH packet to remote host and receive an RPATHACK packet in return.

PROCEDURE di_rcv_xpath;

Receive an XPATH packet from remote host and send an XPATHACK packet in return.

```
PROCEDURE di_rcv_rpath;
```

Receive an RPATH packet from remote host and send an RPATHACK packet in return.

```
PROCEDURE di_snd_escape;
```

Send an ESCAPE packet to remote host and receive an ESCAPEACK packet in return.

```
PROCEDURE di_rcv_escape;
```

Receive an ESCAPE packet to remote host and send an ESCAPEACK packet in return.

```
PROCEDURE di_packet_read (VAR packet: vstring;
                          typ: di_packet_type;
                          timelimit: integer;
                          checkeof: boolean;
                          expected_length: integer);
```

Read a packet from the remote host. 'typ' indicates the type of packet expected. The program is aborted if the packet is not read within 'timelimit' seconds. If 'checkeof' is true, the packet must have the "end-of-segment" bit set. If 'expected_length' is non-zero, then the packet must contain exactly 'expected_length' characters. The packet is returned in 'packet'. All correctly received packets are automatically acknowledged by di_packet_read.

```
PROCEDURE di_packet_convert (packet: vstring;
                             buf: vstring);
```

Converts all escaped character in 'packet' to the correct untranslated character and places the result in 'buf'. The packet header is also removed.

```
PROCEDURE di_read_record (VAR buffer: vstring);
```

Reads packets from the remote host until one is read with the "end-of-segment" flag set. The concatenation of all the packet bodies is returned in 'buffer'.

```
PROCEDURE di_packet_encode (VAR packet: vstring);
```

Escapes all characters in 'packet' that cannot be directly transmitted to the remote host.

```
PROCEDURE di_packet_write (packet: vstring;
                           packlen: integer;
                           typ: di_packet_type;
                           endofrecbit: di_bit;
                           ackwait: integer);
```

Transmits a packet to the remote host. The packet is in 'packet' and contains 'packlen' characters. Its type is filled in from 'typ', the "end-of-segment" flag is set if 'endofrecbit' is true, and the checksum is calculated and inserted into the packet header. After writing the packet, di_write_packet will wait up to 'ackwait' seconds for an acknowledgement. If the time expires with no acknowledgement, the packet is retransmitted. Several retries are attempted.

```
PROCEDURE di_transchar (VAR packet: vstring; ch: char;
                        VAR bytelen: integer);
```

Buffers one character, 'ch' for transmission to the remote host. The character is escaped if necessary. 'packet' contains the packet that is being built by successive calls to di_transchar. When 'packet' becomes full, it is transmitted using di_packet_write. 'bytelen' is incremented by the number of characters buffered.

APPENDIX G

CONFIGURATION FILE FORMAT

The configuration file contains one entry for each defined channel. Entries are separated by one blank line. The first line of an entry contains the one letter channel code for the channel. The second line contains the official host name. The third and subsequent lines may contain synonyms for the official host name and host name/official host name pairs for the hosts whose mail is relayed using this channel.

APPENDIX H

MESSAGE FILE FORMAT

The form of the filenames of messages files is:

<chancode><process-id><time>.<retry-count>

where, <chancode> is the one letter channel code of the messages channel, <process-id> a 4-digit representation of the process identification or job number of the process that submitted the message, <time> is the low order 4 hexadecimal digits of the system time and <retry-count> is normally "00" but can take on other values if needed in order to ensure the uniqueness of a filename.

Each message file is in the following format:

```
m;return address of sender of the message
address of recipient 1
address of recipient 2
        .
        .
        .
^A^A (two control-A's)
message line 1
message line 2
        .
        .
        .
^A^A^A^A^A (five control-A's)
```

The "end-of-text" sentinel (five control-A's) was included so that partial written messages (possibly resulting from system crashes) could be easily detected.

APNDIX I

RFC 822 DRESS PARSER

A BNF description of an C 822 standard address list follows:

```
<Address-list>   ->    <dress>
                 ->    <dress> , <Address-list>

<Address>        ->    <ilbox>
                 ->    <oup>

<Group>          ->    <rase> : <Mailbox-list> ;
                 ->    <rase> : ;

<Mailbox-list>   ->    <ilbox>
                 ->    <ilbox> , <Mailbox-list>

<Mailbox>        ->   <Adr-spec>
                 ->   <Plase> <Route-addr>

<Route-addr>     ->   < ddr-spec> >
                 ->   < oute-list> : <Addr-spec> >

<Route-list>     ->   <Rte>
                 ->   <Rte> , <Route-list>

<Route>          ->   @ omain>

<Addr-spec>      ->   <Lcal-part> @ <Domain>

<Local-part>     ->   <Wod>
                 ->   <Lcal-part> . <Word>
```

```
<Domain>          ->    <Sub-domain>
                  ->    <Sub-domain> . <Domain>

<Sub-domain>      ->    ATOM

<Phrase>          ->    <Word>
                  ->    <Phrase> <Word>

<Word>            ->    ATOM
                  ->    QUOTEDSTRING
```

An ATOM is a string of characters not containing blanks or special characters such as periods, colons, semicolons, angle brackets, quotes, and commas.

A QUOTEDSTRING is a sequence of ASCII characters enclosed in double quotes.

The parser is defined as follows:

```
FUNCTION ap_parse (VAR string: bigvstring):
                  ap_address_list_ref;
```

Parses 'string' into its component fields according to the RFC 822 standard. NIL is returned if the address is illegal.

The data structure returned by 'ap_parse' is defined as follows:

```
ap_addr_kind = (ap_mailbox_kind,ap_group_kind);

ap_mbx_kind = (ap_simple_kind,ap_full_kind);

ap_address_list_ref    = ^ap_address_list;
ap_address_ref         = ^ap_address;
ap_mailbox_ref         = ^ap_mailbox;
ap_mailbox_list_ref    = ^ap_mailbox_list;
ap_group_ref           = ^ap_group;
ap_route_addr_ref      = ^ap_route_addr;
ap_route_list_ref      = ^ap_route_list;
ap_domain_ref          = ^ap_domain;
ap_addr_spec_ref       = ^ap_addr_spec;
```

```
ap_address_list = RECD
  next: ap_address_li_ref;
  elem: ap_address_re
  strpos: integer;     (* position of this address *)
  END;                 (* in the original string   *)

ap_address = RECORD
  CASE kind: ap_addr_knd OF
    ap_mailbox_kind: (bx: ap_mailbox_ref);
    ap_group_kind:   (rp: ap_group_ref);
    END;

ap_mailbox = RECORD
CASE kind: ap_mbx_kindOF
  ap_simple_kind: (adc: ap_addr_spec_ref);
  ap_full_kind:   (nam: vstring_ref;
  fulladdr: ap_route_ddr_ref);
  END;

ap_mailbox_list = RECD
  next: ap_mailbox_li_ref;
  mbox: ap_mailbox_re
  END;

ap_group = RECORD
  name: vstring_ref;
  list: ap_mailbox_lis_ref;
  END;

ap_route_addr = RECORD
  routes: ap_route_lis_ref;
  addr: ap_addr_spec_rf;
  END;

ap_route_list = RECORD
  next: ap_route_list_ef;
  elem: ap_domain_ref;
  END;

ap_addr_spec = RECORD
  local: vstring_ref;
  dmn:   ap_domain_ref;
  END;

ap_domain = RECORD
  next: ap_domain_ref;
  elem: vstring_ref;
  END;
```