         PCMAIL: A Distributed Mail System for Personal Computers

                         Table of Contents

1. Status of this Memo

   This RFC is a discussion of the Pcmail workstation based distributed
   mail system. It is identical to the discussion in RFC-993, save that
   a new, much simpler mail transport protocol is described. The new
   transport protocol is the result of continued research into ease of
   protocol implementation and use issues. Distribution of this memo is
   unlimited.

2. Introduction

   Pcmail is a distributed mail system providing mail service to an
   arbitrary number of users, each of whom owns one or more
   workstations. Pcmail's motivation is to provide very flexible mail
   service to a wide variety of different workstations, ranging in power
   from small, resource-limited machines like IBM PCs to resource-rich
   (where "resources" are primarily processor speed and disk space)
   machines like Suns or Microvaxes. It attempts to provide limited
   service to resource-limited workstations while still providing full
   service to resource-rich machines. It is intended to work well with
   machines only infrequently connected to a network as well as machines
   permanently connected to a network. It is also designed to offer
   diskless workstations full mail service.

   The system is divided into two halves. The first consists of a
   single entity called the "repository". The repository is a storage
   center for incoming mail. Mail for a Pcmail user can arrive
   externally from the Internet or internally from other repository
   users. The repository also maintains a stable copy of each user's
   mail state (this will hereafter be referred to as the user's "global
   mail state"). The repository is therefore typically a computer with
   a large amount of disk storage.

   The second half of Pcmail consists of one or more "clients". Each
   Pcmail user may have an arbitrary number of clients, typically
   single-user workstations. The clients provide a user with a friendly
   means of accessing the user's global mail state over a network. In
   order to make the interaction between the repository and a user's
   clients more efficient, each client maintains a local copy of its
   user's global mail state, called the "local mail state". It is
   assumed that clients, possibly being small personal computers, may
   not always have access to a network (and therefore to the global mail
   state in the repository). This means that the local and global mail
   states may not be identical all the time, making synchronization
   between local and global mail states necessary.

   Clients communicate with the repository via the Distributed Mail
   System Protocol (DMSP); the specification for this protocol appears
   in appendix A. The repository is therefore a DMSP server in addition
   to a mail end-site and storage facility. DMSP provides a complete
   set of mail manipulation operations ("send a message", "delete a
   message", "print a message", etc.). DMSP also provides special
   operations to allow easy synchronization between a user's global mail
   state and his clients' local mail states. Particular attention has
   been paid to the way in which DMSP operations act on a user's mail
   state. All DMSP operations are failure-atomic (that is, they are
   guaranteed either to succeed completely, or leave the user's mail

state unchanged). A client can be abruptly disconnected from the
repository without leaving inconsistent or damaged mail states.

Pcmail's design has been directed by the characteristics of currently
available workstations. Some workstations are fairly portable, and
can be packed up and moved in the back seat of an automobile. A few
are truly portable--about the size of a briefcase--and battery-
powered. Some workstations have constant access to a high-speed
local-area network; pcmail should allow for "on-line" mail delivery
for these machines while at the same time providing "batch" mail
delivery for other workstations that are not always connected to a
network. Portable and semi-portable workstations tend to be
resource-poor. A typical IBM PC has a small amount (typically less
than one megabyte) of main memory and little in the way of mass
storage (floppy-disk drives that can access perhaps 360 kilobytes of
data). Pcmail must be able to provide machines like this with
adequate mail service without hampering its performance on more
resource-rich workstations. Finally, all workstations have some
common characteristics that Pcmail should take advantage of. For
instance, workstations are fairly inexpensive compared to the various
time-shared systems that most people use for mail service. This
means that people may own more than one workstation, perhaps putting
a Microvax in an office and an IBM PC at home.

Pcmail's design reflects the differing characteristics of the various
workstations. Since one person can own several workstations, Pcmail
allows users multiple access points to their mail state. Each Pcmail
user can have several client workstations, each of which can access
the user's mail by communicating with the repository over a network.
The clients all maintain local copies of the user's global mail
state, and synchronize the local and global states using DMSP.

It is also possible that some workstations will only infrequently be
connected to a network (and thus be able to communicate with the
repository). The Pcmail design therefore allows two modes of
communication between repository and client. "Interactive mode" is
used when the client is always connected to the network. Any changes
to the client's local mail state are immediately also made to the
repository's global mail state, and any incoming mail is immediately
transmitted from repository to client. "Batch mode" is used by
clients that have infrequent access to the repository. Users
manipulate the client's local mail state, queueing the changes
locally. When the client is next connected to the repository, the
changes are executed, and the client's local mail state is
synchronized with the repository's global mail state.

Finally, the Pcmail design minimizes the effect of using a resource-
poor workstation as a client. Mail messages are split into two

parts: a "descriptor" and a "body". The descriptor is a capsule
message summary whose length (typically about 100 bytes) is
independent of the actual message length. The body is the actual
message text, including an RFC-822 standard message header. While
the client may not have enough storage to hold a complete set of
messages, it can usually hold a complete set of descriptors, thus
providing the user with at least a summary of his mail state. For
clients with extremely limited resources, Pcmail allows the storage
of partial sets of descriptors. Although this means the user does
not have a complete local mail state, he can at least look at
summaries of some messages. In the cases where the client cannot
immediately store message bodies, it can always pull them over from
the repository as storage becomes available.

The remainder of this document is broken up into sections discussing
the following:

   - The repository architecture

   - DMSP, its operations, and motivation for its design

   - The client architecture

   - A typical DMSP session between the repository and a
     client

   - The current Pcmail implementation

   - Appendices describing the DMSP protocol in detail

3. Repository architecture

   A typical machine running repository code has a relatively powerful
   processor and a large amount of disk storage. It must also be a
   permanent network site, for two reasons. First, clients communicate
   with the repository over a network, and rely on the repository's
   being available at any time. Second, people sending mail to
   repository users rely on the repository's being available to receive
   mail at any time.

   The repository must perform several tasks. First, and most
   importantly, the repository must efficiently manage a potentially
   large number of users and their mail states. Mail must be reliably
   stored in a manner that makes it easy for multiple clients to access
   the global mail state and synchronize their local mail states with
   the global state. Since a large category of electronic mail is
   represented by bulletin boards (bboards), the repository should
   efficiently manage bboard mail, using a minimum of storage to store

bboard messages in a manner that still allows any user subscribing to
the bboard to read the mail. Second, the repository must be able to
communicate efficiently with its clients. The protocol used to
communicate between repository and client must be reliable and must
provide operations that (1) allow typical mail manipulation, and (2)
support Pcmail's distributed nature by allowing efficient
synchronization between local and global mail states. Third, the
repository must be able to process mail from sources outside the
repository's own user community (a primary outside source is the
Internet). Internet mail will arrive with a NIC RFC-822 standard
message header; the recipient names in the message must be properly
translated from the RFC-822 namespace into the repository's
namespace.

3.1. Management of user mail state

Pcmail divides the world into a community of users. Each user is
associated with a user object. A user object consists of a unique
name, a password (which the user's clients use to authenticate
themselves to the repository before manipulating a global mail
state), a list of "client objects" describing those clients belonging
to the user, a list of "subscription objects", and a list of "mailbox
objects".

A client object consists of a unique name and a status. A user has
one client object for every client he owns; a client cannot
communicate with the repository unless it has a corresponding client
object in a user's client list. Client objects therefore serve as a
means of identifying valid clients to the repository. Client objects
also allow the repository to manage local and global mail state
synchronization; the repository associates with every client an
"update list" of message state changes which have occurred since the
client's last synchronization.

A client's status is either "active" or "inactive". The repository
defines inactive clients as those clients which have not connected to
the repository within a set time period (one week in the current
repository implementation). When a previously-inactive client does
connect to the repository, the repository notifies the client that it
has been inactive for some time and should "reset" itself. Resetting
a client has the effect of placing every message in every mailbox
onto the client's update list. This allows the client to get a fresh
global mail state from the repository when it next synchronizes (see
synchronization discussion following). The reset is performed on the
assumption that enough global state changes occur in a week that the
client would spend too much time performing an ordinary local state-
global state synchronization.

Messages are stored in mailboxes. Users can have any number of
mailboxes, which serve both to store and to categorize messages. A
mailbox object both names a mailbox and describes its contents.
Mailboxes are identified by a unique name; their contents are
described by three numeric values. The first is the total number of
messages in the mailbox, the second is the total number of unseen
messages (messages that have never been seen by the user via any
client) in the mailbox, and the third is the mailbox's next available
message unique identifier (UID). The above information is stored in
the mailbox object to allow clients to get a summary of a mailbox's
contents without having to read all the messages within the mailbox.

Some mailboxes are special, in that other users may read the messages
stored in them. These mailboxes are called "bulletin board
mailboxes" or "bboard mailboxes". The repository uses bboard
mailboxes to store bboard mail. Bboard mailboxes differ from
ordinary mailboxes in the following ways:

   - Their names are unique across the entire repository;
     for instance, only one bboard mailbox named "sf-lovers"
     may exist in the entire repository community. This
     does not preclude other users from having an ordinary
     mailbox named "sf-lovers".

   - Subscribers to the bboard are granted read-only access
     to the messages in the bboard mailbox. The bboard
     mailbox's owner (typically the system manager) has
     read/update/delete access to the mailbox.

A bboard subscriber keeps track of the messages he has looked at via
a subscription object. The subscription object contains the name of
the bboard, its owner (the user who owns the bboard mailbox where all
the messages are stored), and the UID of the first message not yet
seen by the subscriber.

Users gain read-only access to a bboard by creating a subscription to
it; they lose that access when they delete that subscription. A list
of all bboard mailboxes available for subscription can be transmitted
to the user on demand.

Associated with each mailbox are any number of message objects. Each
message is broken into two parts--a "descriptor", which contains a
summary of useful information about the message, and a "body", which
is the message text itself, including its NIC RFC-822 message header.
Each message is assigned a monotonically increasing UID based on the
owning mailbox's next available UID. Each mailbox has its own set of
UIDs which, together with the mailbox name and user name, uniquely
identify the message within the repository. A descriptor holds the

following information: the message UID, the message size in bytes
and lines, four "useful" message header fields (the "date:", "to:",
"from:", and "subject:" fields), and sixteen flags. These flags are
given identifying numbers 0 through 15. Eight of these flags have
well-known definitions and are reserved for the repository's use.
The eight repository-defined flags mark:

   - (#0) whether the message has been deleted

   - (#1) whether it has been seen

   - (#2) whether it has been forwarded to the user

   - (#3) whether it has been forwarded by the user

   - (#4) whether it has been filed (written to a text file
     outside the repository)

   - (#5) whether it has been printed (locally or remotely)

   - (#6) whether it has been replied to

   - (#7) whether it has been copied to another mailbox

The remaining eight flags are availble for user use. Descriptors
serve as an efficient means for clients to get message information
without having to waste time retrieving the entire message from the
repository.

3.2. Repository-to-RFC-822 name translation

"Address objects" provide the repository with a means for translating
the RFC-822-style mail addresses in Internet messages into repository
names. The repository provides its own namespace for message
identification. Any message is uniquely identified by the triple
(user-name, mailbox-name, message-UID). Any mailbox is uniquely
identified by the pair (user-name, mailbox-name). In order to
translate between RFC-822-style mail addresses and repository names,
the repository maintains a list of address objects. Each address
object is an association between an RFC-822-style address and a
(user-name, mailbox-name) pair. When mail arrives from the Internet,
the repository can use the address object list to translate the
recipients into (user-name, mailbox-name) pairs and route the message
correctly.

4. Communication between repository and client: DMSP

   The Distributed Mail System Protocol (DMSP) defines and manipulates
   the objects mentioned in the previous section. It has been designed
   to work with Pcmail's singlerepository/multiple-client model of the
   world. In addition to providing typical mail manipulation functions,
   DMSP provides functions that allow easy synchronization of global and
   local mail states.

   DMSP has been completely re-specified in this version of Pcmail.
   Formerly, DMSP was implemented on top of the USP remote-procedure-
   call protocol. Since this protocol is not fully unofficially
   specified (let alone officially specified) anywhere, implementation
   of USP is difficult for sites wishing to implement Pcmail on
   different systems. We therefore have decided to completely redesign
   DMSP. It is now a very simple request/response protocol similar to
   SMTP or NNTP, running directly on a reliable bidirectional byte-
   stream such as TCP. The TCP contact port for DMSP has been
   designated 158. Requests and responses consist of ASCII characters;
   on octet-based transmission streams, each character is transmitted
   rightjustified in an octet with the high-order bit cleared to zero.

4.1. DMSP commands

   DMSP operations consist of an operation name, followed by zero or
   more tab or space characters, followed by zero or more arguments,
   each of which is separated from the operation name and other
   arguments by one or more space or tab characters. All operation
   requests, as well as all responses, must be terminated with a
   carriage-return plus line-feed (CR-LF) pair. All operation names and
   arguments must be taken from the set of alphanumeric characters plus
   the characters dash ("-"), underscore ("_"), and period (".").

   DMSP operation names are case-insensitive; they may be transmitted in
   any combination of upper and lower case. DMSP arguments are case-
   insensitive but case-preserving; in other words a mailbox named
   "MarkL" may be referred to by an operation argument "markl", but will
   always be stored, and transmitted in a repository response, as
   "MarkL"; furthermore, any attempt to create a new mailbox "MaRkL"
   will not be permitted.

   Each operation argument may contain no more than 64 characters. No
   single request or response line may contain more than 512 characters,
   including all white space and the terminating CR-LF.

4.2. DMSP responses

   A DMSP operation always results in a response, which may be followed

in turn by a list, consisting of zero or more lines of CR-LF-
terminated text terminated by a single period (".") plus a CR-LF. A
response is always prefaced by a three-digit reply code; possible
text following the response code can be in any format. The response
code is sufficient to determine whether the operation succeeded or
failed, or whether more text is forthcoming following the response
line. Any text following the response code is for information only,
and need not follow any particular format.

The first digit indicates whether the operation succeeded or failed,
and if it succeeded whether or not more text should be presented to
the repository. Definitions of the first digit are similar to those
of NNTP:

    1XX   Informative message

    2XX   Operation completed successfully

    3XX   Operation completed successfully, present
                   remainder of text and terminate with a single
                   period plus CR-LF pair.

    4XX   Operation was performed and failed for some
                   reason.

    5XX   Operation could not be performed because of a
                   protocol syntax error of some sort.

The second digit indicates the type of object referred to by the
response.

    X0X   Miscellaneous

    X1X   User operation

    X2X   Client operation

    X3X   Mailbox operation

    X4X   Subscription operation

    X5X   Message operation

    X6X   Address operation

In an error response, the final digit can describe the type of error
that occurred. Otherwise, it simply gives a response a unique
number. Numbers 0 through 3 are significant in 4XX-class (error)
responses only. Numbers 0-9 in all other responses serve only to
differentiate responses dealing with the same type of object under
different circumstances.

    4X0   Operation failed because object exists

    4X1   Operation failed because object does not exist

    4X2   Operation failed because of an internal error

    4X3   Operation failed because of an argument syntax
                        error

Each operation generates one of a set of responses, detailed in the
protocol specification appendix.

List termination is determined solely by a well-known character
sequence (CR-LF, period, CR-LF). Since application data could well
accidentally contain this termination sequence, the transmitting
protocol module must modify application data so it contains no
termination sequences. The receiving module must similarly undo the
modification before presenting the data to the application at the
receiving end.

The transmitting module modifies application data as follows: If a
line of application data begins with a period, that period is
duplicated. Since the termination sequence is a single period,
accidental termination has now been prevented.

The receiving protocol checks incoming all incoming data lines for a
leading period. A single period is a list terminator; a period
followed by other text is removed before being presented to the

receiving application.

4.3. DMSP sessions

   A DMSP session proceeds as follows: a client begins the session with
   the repository by opening a connection to the repository's machine.
   The client then authenticates both itself and its user to the
   repository with a "login" operation. If the authentication is
   successful, the user performs an arbitrary number of DMSP operations
   before ending the session with a "logout" operation, at which time
   the connection is closed by the repository.

   Because DMSP can manipulate a pair of mail states (local and global)
   at once, it is extremely important that all DMSP operations are
   failure-atomic. Failure of any DMSP operation must leave both states
   in a consistent, known state. For this reason, a DMSP operation is
   defined to have failed unless an explicit acknowledgement is received
   by the operation initiator. This acknowledgement consists of a
   response code possibly followed by information, as described above.

   Following is a general discussion of all the DMSP operations. The
   operations are broken down by type: general operations, user
   operations, client operations, mailbox operations, address
   operations, subscription operations, and message operations.
   Detailed operation specifications appear at the end of this document.

4.4. General operations

   The first group of DMSP operations perform general functions that
   operate on no one particular class of object. DMSP has three general
   operations which provide the following services:

   In order to prevent protocol version skew between clients and the
   repository, DMSP provides a "send-version" operation. The client
   supplies its DMSP version number as an argument; the operation
   succeeds if the supplied version number matches the repository's DMSP
   version number. It fails if the two version numbers do not match.
   The version number is a natural number like "100", "101", "200". The
   "send-version" operation should be the first that a client sends to
   the repository, since no other operation may work correctly if the
   client and repository are using different versions of DMSP.

   Users can send mail to other users via the "send-message" operation.
   The message must have an Internet-style header as defined by NIC
   RFC-822. The repository takes the message and distributes it to the
   mailboxes specified by the message header's destination fields. If
   one or more of the mailboxes exists outside the repository's user
   community, the repository is responsible for handing the message to a

local SMTP server. The message envelope is generated by the
repository from the message contents since it may be difficult for
some clients to perform envelope-generation functions such as address
verification and syntax checking.

A success acknowledgement is sent from the repository only if (1) the
entire message was successfully transmitted from client to
repository, and (2) the message header was properly formatted. Once
the repository has successfully received the message from the client,
any subsequent errors in queueing or delivery must be noted via
return mail to the user.

The last general operation is the "help" operation. The repository
responds to "help" by printing an acknowledgement followed by a list
of supported commands, terminated with a period plus CR-LF. The
information is intended for display and can be in any format as long
as the individual lines of text returned by the repository are CR-
LF-terminated.

4.5. User operations

The next series of DMSP operations manipulates user objects. The
most common of these operations are "login" and "logout". A client
must perform a login operation before being able to access a user's
mail state. A DMSP login operation takes five arguments: (1) the
user's name, (2) the user's password, (3) the name of the client
performing the login, (4) a flag set to 1 if the repository should
create a client object for the client if one does not exist (0 else),
and (5) a flag set to 1 if the client wishes to operate in "batch
mode" and 0 if the client wishes to operate in "interactive" mode.
The last flag value allows the repository to tune internal parameters
for either mode of operation.

The repository can make one of three responses. First, it can make a
success response, indicating successful authentication. Second, it
can make one of several failure responses, indicating failed
authentication. Finally, it can make a special response indicating
that authentication was successful, but that the client has not been
used in over a week. This last response serves as a hint that the
client should consider erasing its local mail state and pulling over
a complete version of the repository's mail state. This is done on
the assumption that so many mail state changes have been made in a
week that it would be inefficient to perform a normal
synchronization.

When a client has completed a session with the repository, it
performs a logout operation. This allows the repository to perform
any necessary cleanup before closing the network connection.

A user can change his password via the "set-password" operation. The
operation works much the same as the UNIX change-password operation,
taking as arguments the user's current password and a desired new
password. If the current password given matches the user's current
password, the user's current password is changed to the new password
given. Because encryption can be difficult to perform on some
resource-poor clients, passwords are transmitted in clear text.
Clearly this is not an acceptable long-term solution, and
alternatives are welcomed.

4.6. Client operations

DMSP provides four operations to manipulate client objects. The
first, "list-clients", tells the repository to send the user's client
list to the requesting client. The list is a series of lines, one
per client, containing the client's name, followed by whitespace,
followed by a status string. The status is either "inactive" or
"active". As with all text responses, the list is terminated with a
period plus CR-LF.

The "create-client" operation allows a user to add a client object to
his list of client objects. Although the login operation duplicates
this functionality via the "create-this- client?" flag, the create-
client operation is a useful means of creating a number of new client
objects while logged into the repository via an existing client. The
create-client operation requires as an argument the name of the
client to create.

The "delete-client" operation removes an existing client object from
a user's client list. The client being removed cannot be in use by
anyone at the time. Delete-client also requires as an argument the
name of the client to delete.

The last client operation, "reset-client", causes the repository to
place all of the messages in all mailboxes onto the named client's
update list. When a client next synchronizes with the repository, it
will end up receiving a list of all descriptors when it requests a
list of changed message descriptors for a particular mailbox. This
is useful for two reasons. First, a client's local mail state could
easily become lost or damaged, especially if it is stored on a floppy
disk. Second, if a client has been marked as inactive by the
repository, the reset-client operation provides a fast way of
resynchronizing with the repository, assuming that so many
differences exist between the local and global mail states that a
normal synchronization would take far too much time.

4.7. Mailbox operations

   DMSP supports seven operations that manipulate mailbox objects.
   First, "list-mailboxes" has the repository send to the requesting
   client information on each mailbox. The repository transmits one
   line of information per mailbox, terminating the list with a period
   plus CR-LF. Each line contains, in order and separated by
   whitespace, the mailbox name, "next available UID", total message
   count, and unseen message count. This operation is useful in
   synchronizing local and global mail states, since it allows a client
   to compare the user's global mailbox list with a client's local
   mailbox list. The list of mailboxes also provides a quick summary of
   each mailbox's contents without having the contents present.

   The "create-mailbox" has the repository create a new mailbox and
   attach it to the user's list of mailboxes. It takes as an argument
   the name of the mailbox to create.

   "Delete-mailbox" removes a mailbox from the user's list of mailboxes.
   All messages within the mailbox are also deleted and permanently
   removed from the system. Any address objects binding the mailbox
   name to RFC-822-style mailbox addresses are also removed from the
   system. Delete-mailbox takes as an argument the name of the mailbox
   to delete.

   "Create-bboard-mailbox" allows a user to create a bboard mailbox.
   The name given as an argument must be unique across the entire
   repository user community. Once the bboard mailbox has been created,
   other users may subscribe to it, using subscription objects to keep
   track of which messages they have read on which bboard mailboxes.

   "Delete-bboard-mailbox" allows a bboard's owner to delete a bboard
   mailbox. Subscribers who attempt to read from a bboard mailbox after
   it has been deleted are told that the bboard no longer exists.
   Again, the operation's argument is the name of the bboard mailbox to
   delete.

   "Reset-mailbox" causes the repository to place all of the messages in
   a named mailbox onto the current client's update list. When the
   client next requests a list of changed message descriptors for this
   mailbox, it will receive a list of all message descriptors in the
   mailbox. This operation is merely a more specific version of the
   reset-client operation (which allows the client to pull over a
   complete copy of the user's global mail state). Its primary use is
   for mailboxes whose contents have accidentally been destroyed
   locally.

   Finally, DMSP has an "expunge-mailbox" operation. Any message can be

deleted and "undeleted" at will, since this simply changes the value
of a flag attached to the message. Deletions are made permanent by
performing an expunge-mailbox operation. The expunge operation
causes the repository to look through a named mailbox, removing from
the system any messages marked "deleted". Expunge-mailbox takes as
an argument the name of the mailbox to expunge.

4.8. Address operations

DMSP provides three operations that allow users to manipulate address
objects. First, the "list-address" operation returns a list of
address objects associated with a particular mailbox. Each address
is transmitted on a separate line terminated by a CR-LF; the list is
terminated with a period plus CR-LF.

The "create-address" operation adds a new address object that
associates a (user-name, mailbox-name) pair with a given RFC-822-
style mailbox address. It takes as arguments the mailbox name and
the address name.

Finally, the "delete-address" operation destroys the address object
binding the given RFC-822-style mail address and the given (user-
name, mailbox-name) pair. Arguments are the address to delete and
the mailbox it belongs to.

4.9. Subscription operations

DMSP provides five subscription operations. The first, "list-
subscriptions", gives the user a list of the bboards he is currently
subscribing to. The list consists of one line of information per
subscription. Each entry contains the following information, in
order:

   - The bulletin board's name

   - The UID of the first message the subscriber has not yet
     seen

   - The number of messages the subscriber has not yet seen

   - The highest message UID in the bulletin board

"List-available-subscriptions" gives the user a list of all bboards
he can subscribe to. The list consists of bboard names, one per
line, terminated by a period plus CR-LF. "Createsubscription" adds a
subscription to the user's list of subscriptions; it takes as an
argument the name of the bboard to subscribe to. "Delete-
subscription" removes a subscription from the list, and takes as an

argument the name of the subscription to remove. Note that this does
not delete the associated bboard mailbox (obviously only the bboard's
owner can do that). It merely removes the user from the list of the
bboard's subscribers. Finally DMSP allows the user to tell the
repository which messages in a bboard he has seen. Every
subscription object contains the UID of the first message the user
has not yet seen; the "reset-subscription" operation updates that
number, insuring that the user sees a given bboard message only once.
Reset-subscription takes as arguments the name of the subscription
and the new UID value.

4.10. Message operations

The most commonly-manipulated Pcmail objects are messages; DMSP
therefore provides special message operations to allow efficient
synchronization, as well as a set of operations to perform standard
message-manipulation functions.

A user may request a series of descriptors with the "fetch-
descriptors" operation. The series is identified by a pair of
message UIDs, representing the lower and upper bounds of the list.
Since UIDs are defined to be monotonically increasing numbers, a pair
of UIDs is sufficient to completely identify the series of
descriptors. If the lower bound UID does not exist, the repository
starts the series with the first message with UID greater than the
lower bound. Similarly, if the upper bound does not exist, the
repository ends the series with the last message with UID less than
the upper bound. If certain UIDs within the series no longer exist,
the repository obviously does not send them. The repository returns
the descriptors in a list with the following format:

If a descriptor has been expunged, the repository transmits two
consecutive lines of information: the word "expunged" on one line,
followed by the message UID on the next line. "Expunged"
notifications are only transmitted in response to a "fetch-changed-
descriptors" command; they are an indication to the client that
someone else has expunged the mailbox and that the client should
remove the local copy of the expunged message.

If a descriptor has not been expunged, it is presented as six
consecutive lines of information: the word "descriptor" on the first
line, followed by a second line containing the message UID, flag
states (see examples following), message length in bytes, and message
length in lines, followed by four lines containing in order the
message "from:" field, "to:" field, "date:" field, and "subject:"
field. The entire list of descriptors is terminated by a period plus
CR-LF; individual descriptors are not specially terminated since the
first line ("expunged" or "descriptor") of a list entry determines

the exact length of the entry (two lines or six lines).

The "fetch-changed-descriptors" operation is intended for use during
state synchronization. Whenever a descriptor changes state (one of
its flags is cleared, for example), the repository notes those
clients which have not yet recorded the change locally. Fetch-
changed-descriptors has the repository send to the client a maximum
of the first N descriptors which have changed since the client's last
synchronization, where N is a number sent by the client. The list
sent begins with the descriptor with lowest UID. Note that the list
of descriptors is only guaranteed to be monotonically increasing for
a given call to "fetch-changed-descriptors"; messages with lower UIDs
may be changed by other clients in between calls to "fetch-
changeddescriptors". "Fetch-changed-descriptors" takes two
arguments: the name of the mailbox to search, and the maximum number
of descriptors for the repository to return.

Once the changed descriptors have been looked at, a user will want to
inform the repository that the current client has recorded the change
locally. The "reset-descriptors" command causes the repository to
mark as "recorded by current client" a given series of descriptors.
The series is identified by a low UID and a high UID. UIDs within
the series that no longer exist are ignored. Arguments are: mailbox
name, low UID in range, and high UID in range.

Whole messages are transmitted from repository to user with the
"fetch-message" operation. The separation of "fetchdescriptors" and
"fetch-message" operations allows clients with small amounts of disk
storage to obtain a small message summary (via "fetch-descriptors" or
"fetch-changed-descriptors") without having to pull over the entire
message. Arguments are mailbox name, followed by message UID.

Frequently, a message may be too large for some clients to store
locally. Users can still look at the message contents via the
"print-message" operation. This operation has the repository send a
copy of the message to a named printer. The printer name need only
have meaning to the particular repository implementation; DMSP
transmits the name only as a means of identification. Arguments are:
mailbox name, followed by message UID, followed by printer
identification.

Copying of one message into another mailbox is accomplished via the
"copy-message" operation. A descriptor list of length one,
containing a descriptor for the copied message, is returned if the
copy operation is successful. This descriptor is required because
the copied message acquires a UID different from the original
message. The client cannot be expected to know which UID has been
assigned the copy, hence the repository's sending a descriptor

containing the UID. Arguments to copy-message are: source mailbox
name, target mailbox name, and source message UID.

Each message has associated with it sixteen flags, as described
earlier. These flags can be set and cleared using the "set-message-
flag" operation. The first eight flags have special meaning to the
repository as described above; the remaining eight are for user use.
Set-message-flag takes four arguments: mailbox name, message UID,
flag number (0 through 15), and desired flag state (0 or 1).

5. Client Architecture

Clients can be any of a number of different workstations; Pcmail's
architecture must therefore take into account the range of
characteristics of these workstations. First, most workstations are
much more affordable than the large computers currently used for mail
service. It is therefore possible that a user may well have more
than one. Second, some workstations are portable and they are not
expected to be constantly tied into a network. Finally, many of the
smaller workstations resource-poor, so they are not expected to be
able to store a significant amount of state information locally. The
following subsections describe the particular parts of Pcmail's
client architecture that address these different characteristics.

5.1. Multiple clients

The fact that Pcmail users may own more than one workstation forms
the rationale for the multiple client model that Pcmail uses. A
Pcmail user may have one client at home, another at an office, and
maybe even a third portable client. Each client maintains a separate
copy of the user's mail state, hence Pcmail's distributed nature.
The notion of separate clients allows Pcmail users to access mail
state from several different locations. Pcmail places no
restrictions on a user's ability to communicate with the repository
from several clients at the same time. Instead, the decision to
allow several clients concurrent access to a user's mail state is
made by the repository implementation.

5.2. Synchronization

Some workstations tend to be small and fairly portable; the
likelihood of their always being connected to a network is relatively
small. This is another reason for each client's maintaining a local
copy of a user's mail state. The user can then manipulate the local
mail state while not connected to the network (and the repository).
This immediately brings up the problem of synchronization between
local and global mail states. The repository is continually in a
position to receive global mail state updates, either in the form of

incoming mail, or in the form of changes from other clients. A
client that is not always connected to the net cannot immediately
receive the global changes. In addition, the client's user can make
his own changes on the local mail state.

Pcmail's architecture allows fast synchronization between client
local mail states and the repository's global mail state. Each
client is identified in the repository by a client object attached to
the user. This object forms the basis for synchronization between
local and global mail states. Some of the less common state changes
include the adding and deleting of user mailboxes and the adding and
deleting of address objects. Synchronization of these changes is
performed via DMSP list operations, which allow clients to compare
their local versions of mailbox and address object lists with the
repository's global version and make any appropriate changes. The
majority of possible changes to a user's mail state are in the form
of changed descriptors. Since most users will have a large number of
messages, and message states will change relatively often, special
attention needs to be paid to message synchronization.

An existing descriptor can be changed in one of three ways: first,
one of its sixteen flag values can be changed (this encompasses the
user's reading an unseen message, deleting a message, printing a
message, etc). Second, a descriptor can be created, either by the
delivery of a new message or by the copying of a message from one
mailbox to another. Finally, a descriptor can be destroyed, via an
"expunge-mailbox" operation.

In the above cases, synchronization is required between the
repository and every client that has not previously noted the change.
To keep track of which clients have noticed a global mail state
change and changed their local states accordingly, each mailbox has
associated with it a list of active clients. Each client has a
(potentially empty) "update list" of messages which have changed
since that client last synchronized.

When a client connects to the repository, it executes a DMSP "fetch-
changed-descriptors" operation. This causes the repository to return
a list of all descriptors on that client's update list. When the
client receives the changed descriptors, it may do one of two things:
if the descriptor is marked "expunged", it can remove the
corresponding message from the local mailbox. If the descriptor is
not expunged, the client can store the descriptor, thus updating the
local mail state. After a changed descriptor has been recorded, the
client uses the DMSP "reset-descriptors" operation to remove
descriptors from its update list. Those descriptors will now not be
sent to the client unless (1) it is explicitly requested via a
"fetch-descriptors" operation, or (2) it changes again.

In this manner, a client can run through its user's mailboxes,
getting all changes, incorporating them into the local mail state,
and marking the changes as recorded.

5.3. Batch operation versus interactive operation

Because of the portable nature of some workstations, they may not
always be connected to a network (and able to communicate with the
repository). Since each client maintains a local mail state, Pcmail
users can manipulate the local state while not connected to the
repository. This is known as "batch" operation, since all changes
are recorded by the client and made to the repository's global state
in a batch, when the client next connects to the repository.
Interactive operation occurs when a client is always connected to the
repository. In interactive mode, changes made to the local mail
state are also immediately made to the global state via DMSP
operations.

In batch mode, interaction between client and repository takes the
following form: the client connects to the repository and sends over
all the changes made by the user to the local mail state. The
repository changes its global mail state accordingly. When all
changes have been processed, the client begins synchronization; this
incorporates newly-arrived mail, as well as mail state changes by
other clients, into the local state.

In interactive mode, since local changes are immediately propagated
to the repository, the first part of batch-type operation is
eliminated. The synchronization process also changes; although one
synchronization is required when the client first opens a connection
to the repository, subsequent synchronizations can be performed
either at the user's request or automatically every so often by the
client.

5.4. Message summaries

Smaller workstations may have little in the way of disk storage.
Clients running on these workstations may never have enough room for
a complete local copy of a user's global mail state. This means that
Pcmail's client architecture must allow user's to obtain a clear
picture of their mail state without having all their messages
present.

Descriptors provide message information without taking up large
amounts of storage. Each descriptor contains a summary of
information on a message. This information includes the message UID,
its length in bytes and lines, its status (contained in the eight
system-defined and eight user-defined flags), and portions of its

RFC-822 header (the "from:", "to:", "date:" and "subject:" fields).
All of this information can be encoded in a small (around 100 bytes)
data structure whose length is independent of the size of the message
it describes.

Most clients should be able to store a complete list of message
descriptors with little problem. This allows a user to get a
complete picture of his mail state without having all his messages
present locally. If a client has extremely limited amounts of disk
storage, it is also possible to get a subset of the descriptors from
the repository. Short messages can reside on the client, along with
the descriptors, and long messages can either be printed via the DMSP
print-message operation, or specially pulled over via the fetch-
message operation.

6. Typical interactive-style client-repository interaction

The following example describes a typical communication session
between the repository and a client mail reader. The client is one
of three belonging to user "Fred". Its name is "office-client", and
since Fred has used the client within the last week, it is marked as
"active". Fred has two mailboxes: "fred" is where all of his
current mail is stored; "archive" is where messages of lasting
importance are kept. The example will run through a simple
synchronization operation. Typically, the synchronization will be
performed by a mail reader as part of a "get new mail" operation.

First Fred's mail reader connects to the repository and receives the
following banner:

    200 Pcmail repository version 3.0.0 ready

In order to access his global mail state, the mail reader must
authenticate Fred to the repository; this is done via the DMSP login
operation:

    login fred fred-password office-client 0 0

This tells the repository that Fred is logging in via "office-
client", and that "office-client" is identified by an existing client
object in Fred's mail state. The first argument to the login
operation is Fred's repository user name. The second argument is
Fred's password. The third argument is the name of the client
communicating with the repository. The fourth argument tells the
repository not to create "office-client" even if it cannot find its
client object. The final argument tells the repository that Fred's
client is not operating in batch mode but rather in interactive mode.

Fred's authentication checks out, so the repository logs him in.

    200 command OK

Now that Fred is logged in, the mail reader performs an initial
synchronization. This process starts with the mail reader's asking
for an up-to-date list of mailboxes:

    list-mailboxes

The repository replies with:

    230 mailbox list follows:
    fred 2313 10 1
    archive 101 100 0
    .

This tells the mail reader that there are two mailboxes, "fred" and
"archive". "Fred" has 10 messages, one of which is unseen. The next
incoming message will be assigned a UID of 2313. "Archive", on the
other hand, has 100 messages, none of which are unseen. The next
message sent to "archive" will be assigned the UID 101. There are no
new mailboxes in the list (if there were, the mail reader would
create them. On the other hand, if some mailboxes in the mail
reader's local list were not in the repository's list, the program
would assume them deleted by another client and delete them locally
as well).

To synchronize, the mail reader need only look at each mailbox's
contents to see if (1) any new mail has arrived, or (2) if Fred
changed any messages on one of his other two clients subsequent to
"office-client"'s last connection to the repository.

The mail reader asks for any changed descriptors via the "fetch-
changed-descriptors" operation. It requests at most ten changed
descriptors since storage is very limited on Fred's workstation.

    fetch-changed-descriptors fred 10

The repository responds with:

    250 descriptor list follows:
    expunged

```
    2101
    expunged
    2104
    descriptor
    2107 1100011100000010 1400 30
    foo@bar.edu (Foo Jones)
    fred@PTT.LCS.MIT.EDU
    Wed, 9 Dec 87 10:43:52 EST
    A typical subject line
    descriptor
    2312 0000000000000000 12232 320
    joe@athena.mit.edu
    fred@PTT.LCS.MIT.EDU
    Thu, 17 Dec 87 18:24:09 PST
    Another typical subject line
    .
```

If a descriptor changed because it was expunged, it is transmitted as
two lines: the word "expunged" on one line, followed by the message
UID on the next line. If one of its flags changed state, or it is a
new message, it is transmitted as six lines: the word "descriptor" on
one line, followed by a line containing the message UID, flags, and
length in bytes and lines, followed by the to, from, date, and
subject fields, each on one line. The flags are transmitted as a
single string of ones and zeroes, a one if the flag is on and a zero
if the flag is off. All 16 flags are always transmitted. Flag
zero's state is the first character in the flag string; flag
fifteen's is the last character in the flag string.

The first two descriptors in the list have been expunged, presumably
by Fred's expunging his mailbox on another client. The mail reader
removes messages 2101 and 2104 from its local copy of mailbox "fred".
The next descriptor in the list is one which Fred marked for deletion
on another client yesterday. The mail reader marks the local version
of the message as deleted. The last descriptor in the list is a new
one. The mail reader adds the descriptor to its local list. Since
all changes to mailbox "fred" have now been recorded locally, the
update list can be reset:

    reset-descriptors fred 1 2312

The repository responds with:

    200 command OK

indicating that it has removed from "office-client"'s update list all

messages in mailbox "fred" with UIDs between 1 and 2312 inclusive (in
this case just two messages). "Fred" has now been synchronized. The
mail reader now turns to Fred's "archive" mailbox and asks for the
first ten changed descriptors.

      fetch-changed-descriptors archive 10

The repository responds with:

      250 descriptor list follows:
      .

The zero-length list tells the mail reader that no descriptors have
been changed in "archive" since its last synchronization. No new
synchronization needs to be performed.

Fred's mail reader is now ready to pull over the new message. The
message is 320 lines long; there might not be sufficient storage on
"office-client" to hold the new message. The mail reader tries
anyway:

      fetch-message fred 2312

The repository begins transmitting the message:

      251 message follows:
      UID: 2312
      From: joe@bar.mit.edu
      To: fred@PTT.LCS.MIT.EDU
      Date: Thu, 17 Dec 87 18:24:09 PST
      Subject: Another typical subject line

      Fred,

      ...

Halfway through the message transmission, Fred's workstation runs out
of disk space. Because all DMSP operations are defined to be
failure-atomic, the portion of the message already transmitted is
destroyed locally and the operation fails. The mail reader informs
Fred that the message cannot be pulled over because of a lack of disk
space. The synchronization process is now finished and Fred can
start reading his mail. The new message that was too big to fit on
"office-client" will be marked "off line"; Fred can use the mail

reader to either remote-print it or delete and expunge other messages
until he has enough space to store the new message.

Since Fred is running in interactive mode, changes he makes to any
messages will immediately be transmitted into DMSP operations and
sent to the repository. Depending on the mail reader implementation,
Fred will either have to execute a "synchronize" command periodically
or the client will synchronize for him automatically every so often.

7. A current Pcmail implementation

   The following section briefly describes a current Pcmail system that
   services a small community of users. The Pcmail repository runs
   under UNIX on a DEC Microvax-II connected to the Internet. The
   clients run on IBM PCs, XTs, and ATs, as well as Sun workstations,
   Microvaxes, and VAX-750s.

7.1. IBM PC client code

   Client code for the IBM machines operates only in batch mode. Users
   make local state changes in a mail reader; the changes are queued
   until the user runs a network client program. The program connects
   to the repository, performs the queued changes, and synchronizes
   local and global mail states. The network client program then
   disconnects from the repository.

   The IBM PC client code has gone through several revisions since the
   first Pcmail RFC was published. What was once a fairly primitive and
   cumbersome system has evolved into a system that makes excellent use
   of the PC's limited resources and provides a fairly powerful, easy-
   to-use mail reader.

   Users access and modify their local mail state via a mail reader
   written in the Epsilon text editor's EEL extension language. Users
   are given a variety of commands to operate on individual messages and
   mailboxes, as well as to compose outgoing mail.

   Synchronization and the processing of queued changes is performed by
   a separate program, which the user runs as desired. The program
   takes any actions queued while operating the mail reader, and
   converts them into DMSP operations. All queued changes are made
   before any synchronization is performed. The program can be invoked
   directly from the mail reader, without having to exit and restart.

   The limitation of IBM PC client operation to batch mode was made
   because of development environment limitations. The mail reader
   cannot work with the network code inside it because of the network
   program architecture. The only solution was to provide a two-part

client, one part of which read the mail and one part of which
interacted with the repository. Although slightly cumbersome, the
two-program setup works quite well.

7.2. UNIX client code

Client code for the Suns, Microvaxes, and VAX-750s runs on 4.2/4.3BSD
UNIX. It is fully interactive, with a powerful mail reader inside
Richard Stallman's GNU-EMACS editor. Since UNIX-based workstations
have a good deal of main memory and disk storage, no effort was made
to lower local mail state size by keeping message descriptors rather
than message text.

The local mail state consists of a number of BABYL-format mailboxes.
The interface is very similar to the RMAIL mail reader already
present in GNU-EMACS.

The mail reader communicates with the repository through network code
implemented in EMACS-LISP. Changes to the local mail state are
immediately made on the repository; although the repository is fast,
there is a small noticeable delay in performing operations over the
network.

There is no provision for automatic synchronization whenever new mail
arrives or old mail is changed by another client. Instead, users
must get any new mail explicitly. A simple "notification" program
runs in the background and wakes up every minute to check for new
mail; when mail arrives, the user executes a command to get the new
mail, synchronizing the mailbox at the same time.

7.3. Repository code

The repository is implemented in C on 4.2/4.3BSD UNIX. Currently it
runs on DEC VAX-750s and Microvaxes, although other repositories will
soon be running on IBM RT machines and Sun workstations. The
repository code is designed to allow several clients belonging to a
particular user to "concurrently" modify the user's state. A locking
scheme prevents one client from modifying mail state while another
client is modifying the same state.

8. Conclusions

Pcmail is now used by a small community of people at the MIT
Laboratory for Computer Science. The repository design works well,
providing an efficient means of storing and maintaining mail state
for several users. Its performance is quite good when up to ten
users are connected; it remains to be seen whether or not the
repository will be efficient at managing the state of ten or a

hundred times that many users. Given sufficient disk storage, it
should be able to, since communication between different users'
clients and the repository is likely to be very asynchronous and
likely to occur in short bursts with long "quiet intervals" in
between as users are busy doing other things.

Members of another research group at LCS are currently working on a
replicated, scalable version of the repository designed to support a
very large community of users with high availability. This
repository also uses DMSP and has successfully communicated with
clients that use the current repository implementation. DMSP
therefore seems to be usable over several flavors of repository
design.

The IBM PC clients are very limited in the way of resources. The
mail reader/editor combination is quite powerful, making local mail
state manipulation fairly easy. Obviously a big performance
enhancement would be to provide a fully interactive client. As it
is, batch-style synchronization is relatively time consuming due to
the low performance of the PCs. The "batch-mode" that the PCs use
tends to be good for those PCs that spend a large percentage of their
time unplugged and away from a network. It is somewhat inconvenient
for those PCs that are always connected to a network and could make
good use of an "interactive-mode" state manipulation.

The UNIX-based clients are more powerful and easier to use than their
PC counterparts. Synchronization is much faster, and there is far
more functionality in the mail reader (having an interface that runs
within GNU-EMACS helps a lot in this respect). Most of those people
using the Pcmail system use the UNIX-based client code.

I. DMSP Protocol Specification

   Following are a list of DMSP operations by object type, together with
   syntax, and possible responses. Some responses may be followed by
   zero or more lines of text, terminated by a single period plus CR-LF
   pair. Only success responses and common error responses are listed;
   a complete list of possible responses follows this appendix.
   Expressions in angle brackets (i.e. <mailbox-name>) are
   metalinguistic variables indicating a general request or response
   form. Operations with arguments have a sample invocation following
   the operation syntax and response.

   General operations:

        HELP
        100 Repository version xxx. Following are supported:
        HELP
        SEND-VERSION
        SEND-MESSAGE
        LOGIN
        LOGOUT
        ...
        FETCH-MESSAGE
        COPY-MESSAGE
        .

        SEND-VERSION <version-number>
        200 Command OK
        500 version skew!

        i.e. SEND-VERSION 230

        SEND-MESSAGE
        350 enter message; end with "."
        To: markl
        From: markl
        Subject: a test message

        this is a test message
        .

Repository responds:

        200 Command OK
        403 message syntax error

    User operations:

        LOGIN <user> <password> <client> <create-p> <batch-p>
        200 Command OK
        221 Client out of date by > 1 week
        404 Bad password
        405 Client <client-name> is locked
        411 No user named <user-name>
        421 Client <client-name> not found

        i.e. LOGIN markl foo random-client-name 1 0

        LOGOUT
        200 Command OK

        SET-PASSWORD <old-password> <new-password>
        200 Command OK
        404 Incorrect old password

        i.e. SET-PASSWORD foo bar

    Client operations:

        LIST-CLIENTS
        220 Client list <name> <status> follows:
        client-1 active
        client-2 inactive
        client-3 active
        ...
        client-foobar active
        .

     Each line of the list contains a client name, followed by
    whitespace, followed by the word "active" or the word "inactive",
    indicating whether or not the client has connected to the repository
    within the last week.

```
     CREATE-CLIENT <client-name>
     200 Command OK
     403 <client-name> is an illegal name
     420 Client <client-name> exists

     i.e. CREATE-CLIENT new-client


     DELETE-CLIENT <client-name>
     200 Command OK
     421 Client <client-name> not found
     405 Client <client-name> is locked

     i.e. DELETE-CLIENT old-client


     RESET-CLIENT <client-name>
     200 Command OK
     421 Client <client-name> not found
     405 Client <client-name> is locked

     i.e. RESET-CLIENT any-old-client


  Mailbox operations:


     LIST-MAILBOXES
     230 Mbox list <name> <high-UID> <#msgs> <#new> follows:
     mailbox-1 2338 8 1
     mailbox-2 59 44 0
     ...
     mailbox-foobar 19 9 0
     .
```

Each line of the list contains a mailbox name, followed by the
mailbox's next available unique identifier, followed by the number of
messages in the mailbox, followed finally by the number of unseen
messages in the mailbox. Unseen messages are those whose descriptors
have flag #1 ("message has been seen") set to zero.

```
     CREATE-MAILBOX <mailbox-name>
     200 Command OK
     403 <mailbox-name> is an illegal name
     430 <mailbox-name> already exists
     440 <mailbox-name> exists as a bboard subscription
```

```
     i.e. CREATE-MAILBOX current-events


     DELETE-MAILBOX <mailbox-name>
     200 Command OK
     431 mailbox <mailbox-name> not found
     440 <mailbox-name> is a bboard; use delete-bboard-mailbox

     i.e. DELETE-MAILBOX income-tax-information


     CREATE-BBOARD-MAILBOX <mailbox-name>
     200 Command OK
     430 a mailbox named <mailbox-name> already exists.
     430 a bboard mailbox named <mailbox-name> already exists.
     403 <mailbox-name> is an illegal name

     i.e. CREATE-BBOARD-MAILBOX sf-lovers


     DELETE-BBOARD-MAILBOX <mailbox-name>
     200 Command OK
     404 not owner of <mailbox-name>
     431 no bboard mailbox named <mailbox-name>

     i.e. DELETE-BBOARD-MAILBOX rec.autos


     RESET-MAILBOX <mailbox-name>
     200 Command OK
     431 mailbox <mailbox-name> not found

     i.e. RESET-MAILBOX british-cars


     EXPUNGE-MAILBOX <mailbox-name>
     200 Command OK
     431 mailbox <mailbox-name> not found

     EXPUNGE-MAILBOX british-cars


   Address operations:


     LIST-ADDRESSES <mailbox-name>
     260 Address list for <mailbox-name> follows:
     address-1
```

```
      address-2
      ...
      address-6
      .

      or

      431 mailbox <mailbox-name> not found

      i.e. LIST-ADDRESSES archive
```

Each line of the list consists solely of one address.

```
      CREATE-ADDRESS <mailbox-name> <address-name>
      200 Command OK
      403 <mailbox-name> is an illegal name
      431 mailbox <mailbox-name> not found
      460 <address-name> already exists

      i.e. CREATE-ADDRESS markl markl-bug-pcmail


      DELETE-ADDRESS <mailbox-name> <address-name>
      200 Command OK
      431 mailbox <mailbox-name> not found
      461 address <address-name> not found

      i.e. DELETE-ADDRESS markl markl-info-cobol
```

   Subscription operations:

```
      LIST-SUBSCRIPTIONS
      240 subscription list follows:
      bboard-1 2573 33 2606
      bboard-2 541 4 545
      ...
      bboard-6 1530 43 1573
      .
```

   Each line of the list consists of a bulletin-board name, followed by
   the UID of the first message which the user has not yet looked at,
   followed by the number of messages in the bulletin-board that the
   user has not yet looked at, followed by the bulletin-board's next
   available unique message identifier.

```
     CREATE-SUBSCRIPTION <bboard-name>
     200 Command OK
     403 <bboard-name> is an illegal name
     430 A mailbox named <bboard-name> already exists
     431 Bboard mailbox <bboard-name> not found
     440 Already subscribing to <bboard-name>

     i.e. CREATE-SUBSCRIPTION sf-lovers


     DELETE-SUBSCRIPTION <bboard-name>
     200 Command OK
     441 Subscription <bboard-name> not found

     i.e. DELETE-SUBSCRIPTION rec.music


     RESET-SUBSCRIPTION <bboard-name> <new-UID>
     200 Command OK
     441 Subscription <bboard-name> not found

     i.e. RESET-SUBSCRIPTION rec.music.gdead 1210


     LIST-AVAILABLE-SUBSCRIPTIONS
     241 All available bboards follow:
     mod.politics
     sfl
     tcp-ip
     forum
     ...
     comp.emacs
     .


     Each line of the list consists solely of one bulletin-board
     name.


  Message operations:


     FETCH-CHANGED-DESCRIPTORS <mailbox-name> <max-to-send>
     250 Descriptor list follows:
     expunged
     2333
     expunged
     2334
```

```
        descriptor
        2337 0001000001110000 481 14
        croaker@ptt.lcs.mit.edu
        fred@anymachine.mit.edu
        Tue, 19 Jan 88 11:10:03 EST
        a typical subject line
        descriptor
        2339 0000000000000000 1457 40
        bob@lcs.mit.edu
        csr-people@ptt.lcs.mit.edu
        Mon, 18 Jan 88 13:08:17 +0000
        another typical subject line
        expunged
        2340
        .

        or

        431 mailbox <mailbox-name> not found

        i.e. FETCH-CHANGED-DESCRIPTORS markl 100
```

Each element of the descriptor list is either two or six lines long.
Descriptors which have been expunged are transmitted as two lines:
the word "expunged" on one line, followed by the message unique
identifier on the next line. Descriptors which still exist are
transmitted as six lines: the word "descriptor" on one line, followed
by a line containing the message unique identifier, flag states
(sixteen characters either one or zero depending on the associated
flag value), followed by the message length in characters, followed
by the message length in lines. The next four lines contain the
message's "from:", "to:", "date:", and "subject:" fields,
respectively. Flag zero's state is the first character in the flag
string; flag fifteen's is the last character in the flag string.

```
        FETCH-DESCRIPTORS <mailbox-name> <low-uid> <high-uid>
        250 Descriptor list follows:
        descriptor
        2337 0001000001110000 481 14
        croaker@ptt.lcs.mit.edu
        fred@anymachine.mit.edu
        Tue, 19 Jan 88 11:10:03 EST
        a typical subject line
        descriptor
        2339 0000000000000000 1457 40
        bob@lcs.mit.edu
        csr-people@ptt.lcs.mit.edu
```

```
     Mon, 18 Jan 88 13:08:17 +0000
     another typical subject line
     .

     or

     431 mailbox <mailbox-name> not found

     i.e. FETCH-DESCRIPTORS british-cars 12 31


     COPY-MESSAGE <src-mailbox> <target-mailbox> <source-UID>
     250 Descriptor list follows:
     descriptor
     2339 0000000000000000 1457 40
     bob@lcs.mit.edu
     csr-people@ptt.lcs.mit.edu
     Mon, 18 Jan 88 13:08:17 +0000
     another typical subject line
     .

     or

     400 cannot copy message onto itself
     431 target mailbox <target-mailbox> not found
     431 source mailbox <source-mailbox> not found
     451 message <source-UID> not found

     i.e. COPY-MESSAGE markl british-cars 2338


     RESET-DESCRIPTORS <mailbox-name> <low-UID> <high-UID>
     200 Command OK
     431 mailbox <mailbox-name> not found

     i.e. RESET-DESCRIPTORS markl 1 10000


     PRINT-MESSAGE <mailbox-name> <UID> <printer-ID>
     200 Command OK
     401 printer <printer-name> not found
     431 mailbox <mailbox-name> not found
     451 message <UID> not found

     i.e. PRINT-MESSAGE markl 2433 pravda
```

```
SET-MESSAGE-FLAG <mailbox-name> <UID> <flagnum> <state>
200 Command OK
431 mailbox <mailbox-name> not found
451 message <UID> not found
500 flag number <flag-number> out of range

i.e. SET-MESSAGE-FLAG british-cars 23 0 1


FETCH-MESSAGE <mailbox-name> <UID>
251 message follows:
From: markl@ptt.lcs.mit.edu
To: markl@ptt.lcs.mit.edu
Date: Sun, 17 Jan 88 11:11:11 EST
Subject: anything

this is a sample of some
message text
.

or

431 Mailbox <mailbox-name> not found
451 message <UID> not found

i.e. FETCH-MESSAGE current-events 495
```

II. Operations by name

       copy-message
       create-address
       create-bboard-mailbox
       create-client
       create-mailbox
       create-subscription
       delete-address
       delete-bboard-mailbox
       delete-client
       delete-mailbox
       delete-subscription
       expunge-mailbox
       fetch-changed-descriptors
       fetch-descriptors
       fetch-message
       help
       list-addresses
       list-available-subscriptions
       list-clients
       list-mailboxes
       list-subscriptions
       login
       logout
       print-message
       reset-client
       reset-descriptors
       reset-mailbox
       reset-subscription
       send-message
       send-version
       set-message-flag
       set-password

III. Responses by number

      100 Pcmail repository version XXX; following are supported
      200 Command OK
      220 Client list <name> <status> follows:
      221 Client out of date by > 1 week
      230 Mailbox list <name> <high UID> <#msgs> <#new> follows:
      240 Subscription list follows:
      250 Descriptor list follows:
      251 Message follows:
      260 Address list follows:
      350 enter message; end with "."
      400 cannot copy message onto itself
      410 already logged in
      420 client <name> already exists
      430 mailbox <name> already exists
      430 bboard mailbox <name> already exists
      440 subscription <name> already exists
      460 address <name> already exists
      411 no user named <name>
      421 client <name> not found
      431 mailbox <name> not found
      441 subscription <name> not found
      451 message <UID> not found
      461 address <name> not found
      402 internal error message
      403 syntax error in outbound message
      404 bad password or permission denied
      405 mail state is temporarily in use by another client
      406 please log in
      500 operation syntax error or illegal argument