To: Distribution

From: Steve Herbst

Subject: New mail commands

Date: May 1, 1974

OVERVIEW

This document is about a mail command called "mail" and a mail command called "send_mail". Both use the same ring 1 mailbox. The first, to be installed soon, simulates the current mail command. In addition, it offers extended mailbox protection and some upwards-compatible improvements. The second uses a different command interface and can accomodate features that mail cannot.

The new mail command will have a subroutine-callable entry mail_ enabling programs to send mail. In time, there will be a true subroutine send_mail_ and another called read_mail_. Both are described near the end of this MTB. A new command, send_mail, will do all its work using send_mail_. Subsequently, the mail command with some new options will use read_mail_ to read. It may eventually use send_mail_ to send.

A secure mail command calls ring 1 primitives that define all user access to a message segment in terms of 7 extended-access bits. "mail" is concerned with the first five, which enable a user to:

(add)	a	-	add a message
(delete)	d	-	delete any message
(read)	r	-	read any message
(own)	o	-	read or delete only a message that he himself sent
(status)	s	-	find out how many messages are in the mailbox

In the proposed implementations, mail and send_message use the same mailbox. A bit associated with each message says whether that message was put there by mail or by send_message. The remaining two extended-access bits affect whether send_message can send a wakeup when it adds a message. The bits are:

(wakeup)	w	-	wakeups of normal urgency allowed
(urgent)	u	-	urgent wakeups allowed

The sender name and the date and time sent are secure. They are gotten automatically by the ring 1 primitive that adds messages, and cannot be read unless the message can be read.

A mailbox is any message segment whose name ends in ".mbx". The default mailbox resides in the owner's default working directory with the name "<Person>.mbx". The acl's on a mailbox pertain equally to all kinds of messages.

THE PRIMITIVES

Gate entries mailbox_$entry called by mail programs are analogous to gate entries message_segment_$entry for the queue message segment facility.

Entries mbx_mseg_$entry in the ring 1 primitive facility which dispatches all work to be done are analogous to entries queue_mseg_$entry for queue message segments. Like the latter entries, they require that a message segment has "rew" access with brackets [1,1,1] on all acl's and has a minimum default acl for *.SysDaemon.*. (A newly created message segment automatically gets "adros" <Person>.<Project>.*.) Unlike the latter entries, they

1) require the suffix ".mbx" on a mailbox, as opposed to ".ms" on a queue message segment.

2) recognize the "w" and "u" extended access bits.

3) add an acl giving "a" extended access to *.*.*.

4) assume the first portion of each message to contain coded information, and perform various checks concerning the type of message that is being processed.

   The message pointer points to a structure defined by an include file:

```
/* BEGIN  Mailbox Message Include File */

dcl mail_format_ptr pointer aligned;

dcl 1 mail_format aligned based (mail_format_ptr),
     2 version fixed bin(17) aligned,
     2 sent_from char(32) aligned,
              /* a terminal ID, program name,
                 installation name, etc. */
     2 switches aligned,
      3 wakeup bit(1) unaligned,
      3 urgent bit(1) unaligned,
      3 has_been_read bit(1) unaligned,
      3 others bit(69) unaligned,
     2 text char(4096) aligned;
```

/* END Mailbox Message Include File */

5) check the text of every message that is added for illegal ASCII characters and replace these with ¢177's.

mbx_mseg_ then calls the same ring 1 procedures that queue_mseg_ calls.

FIRST IMPLEMENTATION OF "mail"

Because most users and many exec_coms rely on the current interfaces to mail and send_message, the new mail command should work the same as the old one. The following differences are necessary:

1) differences related to mailbox_$entries. For example, a new mailbox "<Person>.mbx" replaces the old "mailbox". When the new mail command is installed, the first time a user types "mail", a default mailbox is created. From this moment on, mail is directed to the new mailbox. Typing "old_mail" prints the remaining contents of "mailbox". When a user tries to send mail to someone who has only a "mailbox", mail calls old_mail$old passing a pointer to its argument list. old_mail is just the old mail command with a few modifications. mail types "Input:" whereas old_mail types "Input".

2) Only the number of messages and not the number of lines is printed at the top when reading mail.

3) If someone has "o" extended access but no "r" extended access, mail tells him about his own messages in the mailbox. It says, "Your messages:", or "You have no messages in <path>." This feature is useful when a mailbox is protected in ring 1. However, "o" access also gives someone the right to delete his own messages. Many people would prefer not to have mail removed from their mailboxes, so "o" has been left off the default acl for *.*.*.

4) A special command is needed to delete the mailbox because it is in ring 1. Commands are also needed to create a mailbox with a name other than the default name, to manipulate access and to list access. The entries "mbcr", "mbdl", "mbsa", "mbda" and "mbla" belong in a program called (tentatively) "extended_access_commands". See MTB-064.

(One more difference which is not necessary:)

5) Messages print from earliest to latest instead of the

other way around.

OVERHEAD OF THIS IMPLEMENTATION

Calls into ring 1 make up the added overhead in mail. These are as follows:

1) one call into ring 1 to open the mailbox.

   The segment is initiated in ring 1. The ring 1 primitive keeps a pointer and passes an "index" to ring 4.

2) one call into ring 1 to get the message count.

   See NOTE below.

3) one call into ring 1 per message read.

   Each call to mailbox_$read_index or mailbox_$incremental_read_index returns only one message (the first/last or the next/previous) so as not to keep the message segment locked for a long time.

4) one call into ring 1 per message deleted.

5) one call into ring 1 per message added.

6) one call into ring 1 to close the mailbox.

   This means terminating it in ring 1.

NOTE: 1) and 2) are saved in the case where there is no mail by a call to hcs_$status_ to get the correct bit count of the segment and by a calculation to determine whether all of this is header. The primitives have to keep an accurate bit count on the mailbox.

FUTURE IMPROVEMENTS TO "mail"

These are upwards-compatible extensions to mail.

1) Reverse option

   "-rev" tells mail to print messages from latest to earliest.

2) Deleting single messages

The ring 1 entries read only one message at a time, usually relative to the last message read. The mail command, nevertheless, asks one question at the end and deletes all the messages. To do this it uses an array of message id's. Any one of these id's could be used to delete a single message. If we want to delete only the 5th message, say, we should do so while mail still has the array.

Implementation: The printed message header begins with a number:

"5) From: Sam.Spade 03/12/36 1435.2 mst Thu"

The final query "Delete?" accepts the following answers:

"yes" or "all"	delete all messages printed
i1 ... in	delete the numbered messages
"save i1 <path>"	save message in a segment and query again
or "save <path>"	save all messages in a segment and query again

3) Classes of messages

We can define new types of messages at one bit per classification. mail and send_message use the "wakeup" bit; mail prints only messages for which that bit is off and print_message prints only those for which it is on.

In the future, either program can make use of the "has_been_read" bit to print each message the first time only. mailbox_$read primitives should turn this bit on. If a message is not deleted after it is read, mail never prints the message again unless the user specifies "mail -all".

4) Forwarding

One way to forward mail is simply to create a link to another mailbox. A drawback to this method is that it does not inform the sender of what is happening and why.

Implementation: A bit in the message segment header is called the forwarding bit. When this bit is on (it is turned on by a command), mail looks in a segment called "<Person>.fwd" in the mailbox owner's directory. The first 168 characters of this segment are a path name for forwarding. The rest is a message of explanation, telling the sender where his mail is being forwarded and perhaps giving him a phone number or address where the owner can be reached.

A more sophisticated use of <Person>.fwd has bits at the beginning of the segment telling:

whether to forward mail
whether to save wakeup messages
whether to forward wakeup messages
whether to notify the sender

A less sophisticated method eliminates <Person>.fwd altogether and does not forward. The forwarding bit in this case says print a standard explanation: "<Person> will not be inspecting his mail."

Mail can query the sender as to whether in fact he wants his mail forwarded, and as to whether he wants it put in the intended mailbox anyway.

5) Subroutine entry "mail_"

The call:

```
call mail_(name, project, pointer, length, code);
```

sends one piece of mail to one user. A Daemon or absentee processor can use this call to send a notification when it completes a particular task. As with the mail command, length cannot exceed 36864 bits or 1 page. mail_ is an entry in "mail".

6) Reading mail by sender and so forth

Various data returned by the primitives or residing in the info structure mail_format can be used to select mail for printing. The following options ought to be available when reading mail:

-from	sender ID
-name	Person only
-project	project only
-type	sent_from ID (in mail_format)
-date	on or before date
-time	at or before time on given date or current date
-length	up to number of characters

To implement these options, mail calls read_mail_, which takes similar options. read_mail_ keeps calling read primitives until it has a message that meets its specifications. mail prints that message, keeps its id in the array for deletion, and calls read_mail_ again.

THE "send mail" COMMAND

"send_mail" is a command that sends mail. It is not intended to replace "mail" but to offer features that mail cannot offer because of incompatibility. (For example, mail's ".NL" to send a message is incompatible with edm's ".NL" to edit a message.) The user types:

send_mail arg1 ... argn

where argi is either:

a destination of the form Person.Project
or just Person (see 4 below)
(all destinations are optional -- see 3 below)

one of the following control arguments:

-pn	the next argument is the path name of a segment to be sent. If this option is not used, send_mail takes its input from the console.
-list	see below
-dl	see below

The user will still type "mail" to read mail.

SPECIAL FEATURES OF "send mail"

1) Mailing list option

The control argument "-list" tells send_mail that the next argument is the relative or absolute path name of a mailing list. The mailing list is a sequence of Person.Project's separated by spaces or newlines. If send_mail is unable to send to somebody on the list, it prints a suitable message.

2) Delete option

The control argument "-dl" when a path is given tells send_mail to delete the input segment after it has been sent successfully to all destinations. Use of this option is preferable to an explicit delete statement in an exec_com because the segment is not deleted if for some reason it could not be sent.

3) Calling edm

When send_mail responds with "Input", it is in edm edit mode. All of edm's operations are available in editing. For example, it is possible to write the edited message into a segment by saying "w <path>". Write without a path name is, as usual, not allowed. In order to send the edited message, there is a new command:

```
z arg1 ... argn
```

where arg$\underline{i}$ is a destination. z with no arguments sends the message to the destinations specified in the send_mail command line. Since z takes arguments, conversely, no destination need be specified in the send_mail command line.

q quits out of editing and out of send_mail.

Implementation: send_mail calls a new entry edm_, which may or may not be part of edm. The call is:

```
call edm_(input_ptr, input_length,
          w_entry, q_entry, badcom_entry);
```

where:

input_ptr and input_length define a piece of storage to be edited. send_mail does not use these.

w_entry is an entry which edm_ calls when it gets the command "w <path>", passing a pointer, a length and a path name.

q_entry is an entry which edm_ calls when it gets a quit request.

badcom_entry is called whenever edm_ gets a command which it does not recognize, for example, the z command. edm_ passes the name of the command and a pointer and length defining the edited text. The code in send_mail that handles a z command tries to put the text into 1 page of automatic storage. If the text is too big, it issues a complaint and returns to edm_.

The following is a sample edit run:

```
send_mail Sam.Spade
Input
  Mr. Spade, I desperately need some assistance.
A certain valuble relic has been lost to me and
I am in dangerous competition with a man who,
it appears, will stop at nothing to get it into
his own hands.

.
Edit
1 valuble
A certain valuble relic has been lost to me and
s/valuble/valuable/
A certain valuable relic has been lost to me and
b
.
Input
  I will call you at your office at 9:00.
                     A Client

.
Edit
z Miles.Archer       /* does not send to Spade */
1 his own hands
his own hands.
a Tell me if one million dollars is too much.
w corr3/12/36
z                    /* does not send to Archer */
q

r 1230 3.079 5.700 73
```

4) Sending by name only

send_mail needs a table in which to look up the Person.Project of a user given only the user's name. It would of course be nice for users to add entries freely to such a table, designating as a lookup name not only their registered last name but any nickname, special interest group or subsystem title by which they desire to gather mail. Likewise, a user should have the option of being unlisted in the table.

The drawbacks of a user-maintained table are:

1) send_mail would need a special query when it encountered multiple entries with the same lookup name.

2) the table would quickly grow to enormous size.

3) everybody should have add and delete permission on their own entries and only read on the others, therefore the table would end up being a message

segment with extended access.

A viable alternative to the user-maintained table is a system-created table:

```
>system_control_dir>mail_table
```

made from the URF (User Registration File, which includes PNT entries, users who are no longer registered, ARPANET users, etc.) and residing in ring 4 with r access to *.*.*. Whenever the URF is updated, a new mail_table is made. Each user is allowed only two lookup names, his registered last name and his alias, both of which are unique to him. His entry in mail_table gives a single default mailbox in some directory. A bit in the URF keeps an entry from appearing in mail_table.

When send_mail sees a destination argument without a period, it interprets the argument as a lookup name. If it finds the corresponding entry in mail_table, it sends the message without further ado.

SEND AND READ SUBROUTINES

When send_mail / edm_ gets a z command, it dispatches one or more calls to send_mail_:

```
call send_mail_(name, project, ptr, length,
                option_ptr, code);
```

If project is blank, send_mail_ looks up name in mail_table. Then it sends message char (length / 9) based (ptr), governed by options in an optional structure based on option_ptr. The options describe:

whether to forward

whether to send a wakeup, what kind, and how it should be handled (so send_message can use send_mail_)

how to treat illegal ASCII

what version of message to send

whether to save an unsent message

A call to read_mail_ reads one message, always the "next" message, of which it keeps track using an internal static message id and flag. The call is:

```
          call read_mail_(path, ptr, length,
                          option_ptr, code);
```

path is also kept in internal static, and read_mail_ gets an index the first time it opens the mailbox. When there are no more messages to read, read_mail_ returns an error code.

A structure based on option_ptr contains information about:

the order in which to read

specifications for a message, including name, date, etc. options and the various bits in mail_format

whether to delete and whether to query