To: MTB Distribution

From: Chris Jones

Date: 30 July 1981

Subject: Proposed Message Facility Enhancements

This MTB describes plans for enhancements to the Multics message facility commands and subroutines.

The two chief motivations for the re-implementation of the user ring message facility are that the current implementation is felt to be unmaintainable, and to provide reasonable subroutine interfaces for subsystems.

Many of the ideas in this MTB were formulated originally by Jim Davis.

Comments and questions should be sent to the author:

Christopher L. Jones
Honeywell / CISL
575 Technology Square
Cambridge MA 02139

(617) 492-9337 or HVN 261-9337

or

CLJones.Multics on System M and MIT-Multics

1 Introduction

This MTB is divided into three parts: a description of the current status of the message facility (that is to say, a description of the problem), an overview of the proposed changes (both command changes and subroutine interfaces), and the documentation of same.

The reader is assumed to be familiar with the current message facility. The commands (accept_messages, delete_message, print_messages, and the various versions of send_message) are documented in MPM Commands (AG92). The send_message subroutines are documented in MPM Subroutines (AG93).

2 Current Deficiencies of the Message Facility

The most immediate need for the message facility is to fix the bugs in it. This is complicated by the convoluted control structure of the current message facility. More than one of the known bugs can be attributed to the great complexity of the current implementation.

One of the most glaring shortcomings is the inability to determine the current state of the message facility (e.g. on which mailboxes messages are being accepted, whether alarms are enabled, etc.). Without this ability, subsystems such as Emacs which wish to temporarily intercept messages cannot properly restore the message facility to the state it was in before messages were intercepted. Thus, Emacs does not intercept messages by default because it cannot save and restore the user's message modes. Every Emacs user must explicitly ask Emacs to intercept messages, yet, if this is not done, and a message arrives, the screen is garbled.

Existing message facility interfaces allow for a subsystem to specify a string to be executed by the command processor when a message arrives. This is not sufficient. What is needed is a way to determine the names of all mailboxes on which a message might arrive. The ability to intercept messages on a given mailbox is of no value if one cannot tell what mailboxes are potential sources of messages.

An additional deficiency is that the command string can be executed an arbitrarily large number of times for the same message if the message facility reminder feature is used. The reminder feature (invoked via "am -time") causes all undeleted messages to be printed (or the call string to be executed) every N minutes. There is no way to tell whether the call string is being executed because of a new message or a reminder.

3 Changes Proposed to the Message Facility

3.1 New Features

Changes are proposed here to correct what are thought to be design mistakes in the message facilty interfaces. Most of these are incompatible in some way. Compatible additions are also proposed to make the message facility more powerful, and easier to use.

3.1.1 Incompatible Interface Changes

accept_messages will now print old messages by default. This means that "-print" will be redundant (although retained for compatibility). Without this change, naive users will not see messages sent to them while they were logged out.

The accept_messages command will take a new control argument, -no_print, to suppress processing pending messages.

If the accept_messages command is used with a mailbox that does not exist, and is not the default mailbox, then the user will be queried as to whether to create it. Currently the default mailbox is created (without comment, in brief mode). This MTB proposes that the comment should always be given when a mailbox is created (unless -force is given; see below), and that the user should have a chance to be queried before a nondefault mailbox is created.

The accept_messages command will have a new control argument, -force (-fc). If this control argument is given, the user will not be queried as to whether or not to create the mailbox if it does not exist; the mailbox will be created.

Using the "-time" control argument will cause messages to be saved by implication. The function provided by -time is useless without -hold.

When the alarm feature (am -time) is used, all output will go to user_i/o. Currently, the alarm processor gives the count of messages on error_output, prints the messages on user_i/o. and gives a "start" control order on user_output. This can cause lost wakeups. When a message is received it will still be printed on user_i/o, and all commands will print on user_output.

The numbers assigned to messages will be consecutive per mailbox. Currently they are consecutive per-process - if three messages arrive in mailbox A, then three more in mailbox B, mailbox A contains messages 1,2, and 3, and mailbox B contains messages 4,5 and 6. Having consecutive numbering will be less disturbing in printing messages. It will make it possible to estimate the message number of a message of interest.

If the user is accepting messages on a mailbox that is not the default, then the entryname of the mailbox will be printed when the message is printed. This will reduce confusion about which mailbox a message is from.

3.1.2 Compatible Interface Changes

The maximum size of a call command string will be 512 characters instead of 120. This string is concatenated with the message text (which can be any size), and passed to the command processor.

The header printed when printing messages in a mailbox to which the user lacks "r" extended access (but has "o") will be changed from "Message from yourself" to "You have N message(s) in PATH", where N will be the number of messages, and messages will be pluralized properly, and PATH will be the pathname of the mailbox. This is almost the same as print_mail (which spells out the number). The mail command prints "Your messages" as a header.

The maximum number of messages that can be "held" without error will be raised from 256 to a very large number (larger than the number of messages a mailbox can currently hold).

When the alarm feature is used on a mailbox other than the default, it will give the mailbox entryname in the same line as the count of messages. Currently there is no way to tell what mailbox is being referred to.

3.2 Desirable Extensions

While this MTB was being written users have requested certain additional features which have not been included in the current proposal. All of these ideas have merit and would improve the message facility, and so are documented here for future consideration. The proposals are ranked below in estimated order of difficulty (from most difficult to least difficult).

- Allowing more than one process to receive wakeups from a given mailbox.
- Warning the user when attempting to send a message to a suspended process.
- Making it possible for user ring to determine whether a current process is receiving wakeups on a given mailbox (besides the current kludge of sending a message and testing the returned error code).
- Providing the ability to discriminate between automatic or "computer generated" messages (e.g. answering service messages, mail notifications) and interactive or "human generated" messages.

3.3 Subroutine Interfaces

There are two reasons to add subroutine interfaces to the message facility.

First, subsystems in existence and being planned need to be able to process messages in order to provide new user interfaces to the electronic mail facility. For example, Emacs keeps conversations with separate users in separate buffers, and is able to log both sides of a conversation. Other subsystems are being planned for "conferencing". The second reason to add subroutine interfaces to the message facility is to give greater control over unexpected output to video subsystems (such as Emacs and the Menu System). These environments require total control over all writing on the user's screen. As things stand now, a message arriving is printed by the message facility, and thus unexpectedly alters the contents of the screen. (This problem might be better fixed by providing better control over I/O switches, but that solution is beyond this MTB.) Aside from messages, timers are the only other way that unexpected I/O can occur on the user_i/o switch.

3.4 Documentation Changes

The following pages present new documentation for the MPM Commands description of the accept_messages command and document the new subroutine interfaces. Possibly the subroutine interfaces should be considered "internal", and documented only in a PLM or SDN of some sort, if at all.

The subroutines below are to be declared obsolete, in favor of mail_system_, which provides all of the facilities currently provided by these subroutines. An appropriate note should go into AG93, the MPM Subroutines Guide.

- send_message_
- send_message_$acknowledge
- send_message_$express

3.4.1 accept_messages

The following changes should be made to the description of the accept_messages command:

In the description of the -call control argument, add

The maximum length of cmdline is 512 characters.

Add the following four control arguments. Note that -mail and -no_mail are already implemented, but have not been documented.

-mail, -ml
sets the user's process so that when mail is sent to the mailbox a message will be printed. This is the default.

-no_mail, -nml
prevents mail notifications from being printed when mail arrives in the given mailbox.

-no_print
prevents accept_messages from printing messages that were already in the mailbox when the command was given.

-force, -fc
create the mailbox if it does not exist without querying the user as to whether or not the mailbox should be created.

Add to the description of the -print control arg:

(This is the default)

Add to the description of the -time control argument:

Using this control argument causes messages to be held, even if -hold was not given.

Add the following paragraph at the end of the description:

> It is possible to accept messages on more than one mailbox at a time, and to accept messages on a mailbox other than the default. If a mailbox other than the default is to be used, and it does not exist, then the accept_messages command queries the user as to whether or not it should be created. When messages are printed from a mailbox other than the default, the mailbox is always identified.

3.4.2 message_facility_

----
message_facility_

The message_facility_ subroutines allow a subsystem to control the behavior of the message facility. This is useful, for example, to process messages in a non-standard way.

Name: message_facility_$get_message_info

The message_facility_$get_message_info subroutine returns a pointer to an array of structures containing information about each mailbox used by the message facility in this process. The caller should not write into any of the structures.

Usage

```
     declare message_facility_$get_message_info entry (ptr, ptr,
fixed bin (35));

     call message_facility_$get_message_info (areap, mbxdatap, code);
```

where:

1. areap (input)
   is a pointer to an area in which the information will be returned.

2. mbxdatap (output)
   is a pointer to a structure whose form is given below. If null, then the caller is not accepting messages on any mailbox.

3. code (output)
   is a standard system error code.

Upon return from message_facility_$get_message_info, mbxdatap point to a structure of the form:

```
     dcl     1 mbx_array_struc       aligned based,
               2 version             fixed bin,
               2 n_mbxs              fixed bin,
               2 mbx_array (1 refer mbx_array_struc.n_mbxs) like message_data;
```

The structure describing the mailbox (message_data) has the following format:

```
     dcl     1 message_data          aligned based (mbxdatap),
               2 version             fixed bin,
               2 dn                  char (168),
               2 en                  char (32),
               2 index               fixed bin,
               2 flags,
                 3 short_format      bit (1) unal,
                 3 hold_messages     bit (1) unal,
                 3 notify_mail       bit (1) unal,
                 3 default_mbx       bit (1) unal,
                 3 deferred          bit (1) unal,
                 3 pad               bit (31) unal,
               2 prefix              char (12) var,
               2 alarmtime           fixed bin (71),
               2 last,
                 3 last_message_ptr ptr,
                 3 last_id           bit (72) aligned,
                 3 last_sender       char (32),
               2 last_message        fixed bin,
               2 message_ptr         ptr,
               2 n_stacked           fixed bin,
               2 command             char (512) varying,
               2 wakeup_processor,
                 3 wakeup_handler    entry variable,
                 3 wakeup_closure    ptr,
               2 alarm_processor,
                 3 alarm_handler     entry variable,
                 3 alarm_closure     ptr;
```

where

version	is the version of this structure.
dn	is the name of the directory containing the mailbox.
en	is an entry name (including suffix) of the mailbox.
index	is the index that identifies this mailbox to the ring one message segment primitives.
flags	give the state of this mailbox.
short_format	messages are to be printed in short format.
hold_messages	messages are to be held in the mailbox until explicitly deleted.

notify_mail	mail notifications are to be printed
default_mbx	this mailbox is the default mailbox. This is the mailbox used when no box is explicitly named in commands.
deferred	messages are currently deferred in this mbx.
pad	should not be referenced.
prefix	is a string to be printed before the text of a message. It may contain ioa_ control characters. It is set by the -prefix control argument to accept_messages.
alarmtime	if non 0, indicates that this mailbox is in reminder mode. If there are any messages in the mailbox, an alarm goes off every "alarmtime" seconds, and the alarm handler processes all messages. This mode is set by the -time control argument to accept_messages.
last	contains information about the last message received.
last_message_ptr	is a pointer to a mail_format structure for the last message or null if there is no last message.
last_id	is the unique id of the last message. The time of the last message may be extracted from the message id. (see below)
last_sender	is the Person.Project for the last sender.
last_message	if messages are being held, then it is the index into the message array (see below) of the information about the highest numbered message. Otherwise it is zero.
message_ptr	is a pointer to a structure containing an array of information about all messages currently in the mailbox or received in this process in hold mode. The structure itself is not to be modified by users.
n_stacked	is the number of undeleted messages in the mailbox.
command	is the character string to be passed to the command processor to process the message. This string is set by the -call control argument to accept_messages.
wakeup_processor	is information about the routine used to handle wakeups for this mailbox when a new message arrives. This routine may print the message, invoke the command processor with the call string, or perform some arbitrary task defined by the subsystem writer.
wakeup_handler	is the routine to be called.

wakeup_closure	is a pointer to information supplied by the supplier of the wakeup_handler. Its value is defined by the subsystem writer.
alarm_processor	is information about the routine used to handle mailbox alarms. These alarms are set using the -time control argument to the accept_messages command, and are typically used to remind the user of undeleted messages.
alarm_handler	is the routine to be called for alarms.
alarm_closure	is a pointer to information for the alarm handler routine. The message facility makes no use of this pointer, only passes it to the called routine.

Name: message_facility_$set_call_handlers

The message_facility_$set_call_handlers entry point is used by a subsystem writer to specify handler routines to process messages. One routine is called whenever a new message arrives, the other is called when a special alarm timer goes off. Calling sequences are given below.

Usage

```
     declare  message_facility_$set_call_handlers  entry  (char (*), char (*),
            entry, ptr, entry, ptr, fixed bin (35));

     call message_facility_$set_call_handlers   (mbx_dn,   mbx_en,  wakeup_ev,
            wakeup_info, alarm_ev, alarm_info, info, code);
```

where

mbx_dn (input)
is the name of the directory containing the mailbox.

mbx_en (input)
is the name (including suffix) of the mailbox.

wakeup_ev (input)
is the routine to be called when a new message arrives.

wakeup_info (input)
points to information for the wakeup routine. It may be null. It is passed to the wakeup routine, and the message facility makes no use of it.

alarm_ev (input)
is the routine to be called if a mailbox alarm timer goes off.

alarm_info (input)
points to info for the alarm handling routine.

code (output)
is a standard system error code.

Restrictions on the Message Handlers:

A user should be familiar with interprocess communication in Multics and the pitfalls of writing programs which can run asynchronously within a process. For example, if a program does run asynchronously within a process and it does input or output with the tty_ I/O module, then the program should issue the start control order of tty_ before it returns. This is necessary because a wakeup from tty_ may be intercepted by the asynchronous program.

The Wakeup Routine

The wakeup routine is called when a wakeup is received on a given mailbox. It means a message has arrived. The usual action is to print the message. Other actions might be logging the message or replying to it.

If messages are being held, the message is saved before the call.

After the call, if code is 0, the message is considered processed. Otherwise it remains pending, but no error message is printed. Messages that are not to be retained are deleted from the mailbox after a sucessful call. Information about the message is saved for use by the last_message (and related) commands.

The supplied routine should be declared compatibly with the following usage:

```
     declare wakeup_routine entry (ptr, fixed bin, ptr, ptr, fixed bin (35));

     call wakeup_routine (info, message_no, mail_format_ptr, mbx_data_ptr,
             code);
```

where

info (input)
: is the info pointer supplied in the call to message_facility_$set_call_handler.

message_no (input)
: is 0 if messages are not being held, otherwise it is the message number.

mail_format_ptr (input)
: points to the mail_format structure, which describes the message to be processed. This structure is defined in mail_format.incl.pl1.

mbx_data_ptr (input)
: points to a message_data structure, which describes the mailbox the message was received in. This structure is defined above. The user must not write into this structure.

code (output)
: is set by the called routine.

The time of the last message may be extracted from the unique message id by assigning it to the following structure:

```
     dcl  1 mbx_msg_id_str          aligned based,
            2 unknown               bit (18) unal,
            2 time                  bit (54) unal;
```

where:

unknown	has an undefined value and should not be referenced.
time	is the low order 54 bits of a standard Multics time. To convert it to a standard time use the binary builtin.

The Alarm Handler

The alarm routine handles message facility alarm calls. Alarm calls are set by the -time control argument to the the accept_messages command. The default routine prints a count of the number of undeleted messages, lists the messages, then reschedules the timer. See the documentation of timer_manager_ for a description of timer alarm calls.

The supplied routine should be declared compatibly with the following usage:

```
     declare alarm_routine entry  (ptr, ptr);

     call alarm_routine (alarm_info, mbx_data_ptr);
```

where

alarm_info	(input) is the info pointer supplied when the handler was set.
mbx_data_ptr	(input) points to a message_data structure which describes the mailbox whose alarm timer went off. The user must not write into this structure.

The message_ptr in the message_data structure points to a structure which has the following form (declared in message_data.incl.pl1)

```
     dcl  1 msg_array_struc (message_data.n_stacked) aligned based,
            2 slot            bit (72),
            2 mesno           fixed bin,
            2 printed         bit (1);
```

where:

slot	is the unique message ID of this message.
mesno	is the message number of this message. Messages are numbered consecutively per mailbox, starting with message number 1. The message number of a message is passed to the wakeup routine when it is invoked.
printed	is set to "1"b if this message has been processed (either by a wakeup routine or an alarm routine).

Name: message_facility_$get_call_handlers

The message_facility_$get_call_handlers entry point is used by a subsytem to obtain the current handler routines for message wakeups and alarm clock wakeups for a given mailbox. This entry point may be used before a call to set the handlers to allow the caller to restore the handlers later. The information can also be obtained from the message_facility_$get_message_info entry point, this entry is available for completeness.

Usage

```
     declare message_facility_$get_call_handlers entry (char (*), char (*),
            entry, ptr, entry, ptr, fixed bin (35));

     call message_facility_$get_call_handlers (mbx_dn, mbx_en, wakeup_ev,
            wakeup_info, alarm_ev, alarm_info, code);
```

All parameters have the same meaning as for the set_call_handlers entry point, but, are output, except for the directory name and entry name of the mailbox.

3.4.3 SRB Notice

The message facility has been re-implemented in order to fix all known bugs and provide subroutine interfaces for managed video subsytems.

New control args have been added to the accept_messages command.

The accept_messages command now prints pending messages by default, always creates a mailbox if none exists, and assumes "-hold" if "-time" is given.

The subroutines to send messages (send_message_, send_message_$acknowledge_, send_message_$express) are now obsolete, and will be deleted in a future release. Users should convert to the more general subroutine mail_system_.

Appendix A - Time Requirements

The work outlined in this MTB can comfortably be done by one person. It is estimated that the complete re-implementation of the message facility can be done in two (2) person-months. Once the design of the message facility has been finalized, one person-month of effort would be enough to recode the message facility. This task could be done by one person. Debugging, exposure, and installation would use up the remaining time. It would not be necessary to devote a person full-time to this part of the effort, which could reasonably be completed in two months of real time.