

**HONEYWELL**

**MULTICS EXTENDED**  
**MAIL SYSTEM**  
**USER'S GUIDE**

**SOFTWARE**

# MULTICS EXTENDED MAIL SYSTEM USER'S GUIDE

## SUBJECT

Tutorial Introduction to the Multics Extended Electronic Mail System

## SPECIAL INSTRUCTIONS

Refer to the Preface for "Significant Changes".

This document supersedes Order No. CH23, Revision 0, dated September 1981.

The manual has been extensively revised. Change bars in the margins indicate technical changes and additions; asterisks denote deletions.

This manual assumes basic knowledge of the Multics system provided by the 2-volume set, New Users' Introduction to Multics — Part I Order No. CH24 and Part II Order No. CH25.

## SOFTWARE SUPPORTED

Multics Software Release 10.1

## ORDER NUMBER

CH23-01

February 1983

**Honeywell**

## PREFACE

The purpose of this manual is to help you become familiar with the Multics extended electronic mail system. This manual provides you with an illustrated discussion of the `print_mail` and `read_mail` commands for receiving mail, the `send_mail` command for creating and sending mail, and a large variety of useful requests and control arguments to aid you in utilizing the full capacity of the extended mail system.

Readers are expected to know the Multics concepts and terms described in the 2-volume set, New Users' Introduction to Multics (Order Nos. CH24 and CH25). These two manuals are referred to throughout this manual as the New Users' Intro - Part I and Part II. Also very useful is the Qedx Text Editor Users' Guide (Order No. CG40) which is referred to as the Qedx Users' Guide.

Section 1 of this manual introduces the Multics extended mail system.

Section 2 reviews the `print_mail` command.

Sections 3 and 4 introduce the `send_mail` and `read_mail` commands respectively, detailing the requests and control arguments most useful for novice users.

In Section 5 you learn how to send messages to more than one person, how this affects message headers, and how to make further adjustments yourself to the header information.

The information and specifications in this document are subject to change without notice. This document contains information about Honeywell products or services that may not be available outside the United States. Consult your Honeywell Marketing Representative.

Section 6 demonstrates several requests that the mail system offers for storing mail.

Section 7 suggests a variety of techniques for advanced use of the mail system.

The reference descriptions for the three mail system commands discussed in this manual are found in Appendix A. Mailbox commands are described in Appendix B.

A glossary of the terms introduced in this manual is in Appendix C.

### Manual Conventions

A few conventions and special symbols should be recalled before you begin to explore the Multics mail system.

Throughout the manual, the term "mail system" is used to indicate the "extended electronic mail system".

Terms within angle brackets (<...>) are used to convey the kind of word that you are to provide in the indicated space. For example, <User\_id> means that you are to type a User\_id. Any exceptions to this usage are noted.

Technical or other unfamiliar terms are CAPITALIZED when used for the first time, and are included in the glossary (Appendix E).

In examples, an exclamation point is used to indicate a line that you type at the terminal. You do not type the exclamation point, nor does Multics type it as a way of prompting you. It is strictly a typographical convention, to distinguish between typing done by you and typing done by Multics.

All commands, and most requests and control arguments, have short names. The short names are used in most examples throughout the manual.

Mail system messages are referred to as both "messages" and "mail" in this manual. However, you will also encounter other types of messages as you work on Multics. "Interactive messages" are created by users with the send\_message command. Messages from the Multics operating system are generally called "system notices". "Error messages" are also sent by the operating system, although these messages often begin with the name of the particular command that has been used incorrectly. Here are examples of all three of these types of messages:

```
interactive
message ==> From Lotte.ProjDog 08/01/80 09:03 mst Fri: Hi

system
notice ==> Mail delivered to Mnemosyne.ProjCat.

error
message ==> send_mail: No project name supplied. FNewton
```

### Significant Changes in CH23-01

Information on abbrev processing within the mail system has been added to Section 7.

The first two appendixes of the original manual contained information on interactive messages and the memo command. These have been removed from the current revision. See the Commands manual for descriptions of these topics.

Appendix A, which contains information on the mail system commands, has been extensively revised and has no change bars. It contains many new requests and control arguments.

For purposes of clarity and ease of use, the MPM set has been reorganized. The six former MPM manuals, the Tools manual, and the RCP Users' Guide have been consolidated into a new set of three manuals.

Multics Programmer's Reference Manual (AG91)  
contains all the reference material from the former eight manuals. It is referred to in text as the Programmer's Reference manual.

Multics Commands and Active Functions (AG92)  
contains all the commands and active functions from the former eight manuals. It is referred to in text as the Commands manual.

Multics Subroutines and Input/Output Modules (AG93)  
contains all the subroutines and I/O modules from the  
former eight manuals. It is referred to in text as the  
Subroutines manual.

The following manuals are obsolete:

<u>Name</u>	<u>Order No.</u>
MPM Peripheral Input/Output	AX49
MPM Subsystem Writers' Guide	AK92
Programming Tools	AZ03
MPM Communications I/O	CC92
Resource Control Users' Guide	CT38

## CONTENTS

		Page
Section 1	Introduction . . . . .	1-1
	The Mailbox . . . . .	1-2
	Users With Multiple Projects . . . . .	1-2
	The Message . . . . .	1-3
	Requests (read_mail and send_mail) . . . . .	1-4
	Control Arguments and Requests . . . . .	1-5
	How to Use Your Mailbox . . . . .	1-6
Section 2	The Print_Mail Command . . . . .	2-1
Section 3	The Send-Mail Command . . . . .	3-1
	Basic send_mail Command . . . . .	3-1
	The Request Loop . . . . .	3-2
	Viewing Your Message . . . . .	3-3
	The print Request . . . . .	3-3
	The print_header Request . . . . .	3-4
*	Editing Your Message . . . . .	3-4
	Sending Your Message . . . . .	3-7
	Message Filling . . . . .	3-8
	Quitting . . . . .	3-9
	Assistance . . . . .	3-9
	The ? Request . . . . .	3-10
	The list_requests Request . . . . .	3-10
	The help Request . . . . .	3-11
	The list_help Request . . . . .	3-12
	send_mail Control Arguments . . . . .	3-12
Section 4	The read_mail Command . . . . .	4-1
	Basic read_mail Requests . . . . .	4-1
	Listing and Printing . . . . .	4-3
	The list Request . . . . .	4-3
	The print Request . . . . .	4-4
	Message Specifiers . . . . .	4-4
	Keywords . . . . .	4-4
	Ranges . . . . .	4-5
	print Request Control Arguments . . . . .	4-6
	Replying to Messages . . . . .	4-7
	Forwarding a Message . . . . .	4-8
	Deletion and Retrieval . . . . .	4-8
	The delete Request . . . . .	4-8
	The retrieve Request . . . . .	4-10
	Quitting . . . . .	4-11

CONTENTS (cont)

	Page
Assistance . . . . .	4-12
The ? Request . . . . .	4-12
The list_requests Request . . . . .	4-13
The help Request . . . . .	4-14
The list_help Request . . . . .	4-15
read_mail Command Control Arguments . . . . .	4-15
Section 5   More On Sending a Message . . . . .	5-1
Sending To Several People . . . . .	5-1
The to Request . . . . .	5-1
The send Request . . . . .	5-3
The cc Request . . . . .	5-4
Related Header Modifications . . . . .	5-5
The remove Request . . . . .	5-5
The subject Request . . . . .	5-6
The from Request . . . . .	5-7
The -comment Control Argument . . . . .	5-7
Section 6   Storing Your Mail . . . . .	6-1
Your Logbox . . . . .	6-1
The log Request . . . . .	6-1
Examining Your Logbox . . . . .	6-2
Additional Mailboxes . . . . .	6-3
The save Request . . . . .	6-3
The send Request . . . . .	6-4
Examining Other Mailboxes . . . . .	6-4
Your Saveboxes . . . . .	6-5
Other People's Mailboxes . . . . .	6-5
Mail Segments . . . . .	6-5
The append Request . . . . .	6-6
The write Request . . . . .	6-6
The preface Request . . . . .	6-7
Section 7   Advanced Mail Features . . . . .	7-1
Abbreviations . . . . .	7-1
More on Control Arguments . . . . .	7-3
Control Arguments and start_up.ec Segments . . . . .	7-4
Escaping to Command Level . . . . .	7-5
The .. Escape . . . . .	7-5
Re-entering the Mail System . . . . .	7-6
Active Requests . . . . .	7-7
More Requests . . . . .	7-8
The execute Request . . . . .	7-8
The apply Request . . . . .	7-9
The exec_com Request . . . . .	7-9
Appendix A   Mail System Commands . . . . .	A-1
print_mail (prm) . . . . .	A-2



CONTENTS (cont)

	Page
read_mail (rdm) . . . . .	A-6
send_mail (sdm) . . . . .	A-47
Appendix B Mailbox Commands . . . . .	B-1
mbx_create (mocr) . . . . .	B-3
mbx_delete_acl (mbda) . . . . .	B-4
mbx_list_acl (mbla) . . . . .	B-5
mbx_set_acl (mbsa) . . . . .	B-7
Appendix C Glossary . . . . .	C-1
Index . . . . .	i-1

## SECTION 1

### INTRODUCTION

The Multics extended mail system allows you to receive, send, edit, and save messages in a variety of ways, using a set of three interactive (prompting) commands. The `send_mail` command enables you to send mail to as many recipients as you want, with the option of changing the elements of the message, such as who the message is to and from, what the title is, and the text of the message. A choice of two commands, `read_mail` or `print_mail`, lets you manipulate your incoming messages with either a complete and versatile mail processing system or a simple subset of this system, respectively.

The `read_mail` and `send_mail` commands are complementary; although their primary tasks are different, they share several functions. For example, each command has access to a group of internal mail system info segments explaining `read_mail` and `send_mail` requests. The two commands also have many similar requests and control arguments. This can seem rather confusing at first, but as you read on in this manual and become more familiar with the mail system, you will see that two identical requests are usually part of a feature that is shared by the two commands, and therefore the requests both perform the same action. The manual is organized around the major features of the mail system and their related requests, in order to clarify these relationships.

## THE MAILBOX

You must have a mailbox to be able to receive messages. The mail system automatically creates a permanent mailbox for you, the first time you issue either the `print_mail` or the `read_mail` command. (This mailbox can also be created by issuing the `accept_messages` or `print_messages` commands, because the same mailbox also stores incoming interactive messages.) The pathname for this default mailbox is:

```
>udd>Project_id>Person_id>Person_id.mbx
```

as, for example, in this pathname:

```
>udd>ProjCat>Willow>Willow.mbx
```

for the user Willow registered on the ProjCat project.

Your mailbox is a container for messages, with its own set of extended access modes. Extended access modes provide a specialized form of control, specifying what one can do with individual messages in a mailbox. Full access is granted to you; the default access for other users gives them permission to send messages to your mailbox, and to read and delete only their own messages. You may extend or curtail the access using mailbox ACL commands. Extended access modes and mailbox commands are described in Appendix B.

## Users With Multiple Projects

Some users are registered on more than one project, and could thus have more than one personal mailbox. In this case it is important to create a mailbox in only one of your home directories, and to then make "links" from each other home directory to this mailbox, so that when you are logged in on one project and receive mail at another project, you can get immediate notice of the message and process it without having to log into the other project.

As an example, user Ching is registered on three projects: ProjCat, Doc, and SoftWork. To make links to one of her directories (ProjCat) from the other two, she creates a mailbox in her ProjCat directory, with the pathname:

```
>udd>ProjCat>Ching>Ching.mbx
```

After she has created one mailbox, she logs out and logs into another of her projects (Doc). There she types the link command, followed by the pathname of the mailbox from her first home directory:

```
! link >udd>ProjCat>Ching>Ching.mbx
```

She logs out again, and repeats this from within her third project:

```
! login Ching SoftWork
:
:
r 10:37 1.485 32
! link >udd>ProjCat>Ching>Ching.mbx
```

If she had already created a mailbox in her SoftWork project, the link command would ask her:

```
link: Do you wish to delete the old mailbox
>udd>SoftWork>Ching>Ching.mbx ?
```

She would answer yes to this question, because she wants only one mailbox.

#### THE MESSAGE

Messages all have a common format within the mail system. Each one begins with a header consisting of information about the message. The standard header tells you who wrote the message and to whom it was sent, the date and time the message was sent, and what the subject of the message is. This information is displayed in header fields, one field to a line. Here is an example of a standard header:

```
Date: 1 August 1980 09:14 mst
From: Moch.ProjCat
Subject: picnic
To: Willow.ProjCat
```

The first line, the Date field, informs you of the date and time the message was actually written. The person who wrote the message is noted in the From field, and the title of the message is in the Subject field. The To field lists the person or people who received the message.

The text of the message follows the header, with one blank line between. Here is an example of a complete message:

```
Date: 1 August 1980 09:14 mst
From: Moch.ProjCat
Subject: picnic
To: Willow.ProjCat
```

```
There will be a meeting at 9:30 on Tuesday to discuss
plans for the umpteenth annual office picnic.
Everyone is asked to attend -- please inform
the others in your project.
```

As incoming mail, the entire message can be read, kept, or deleted using the `print_mail` command. Within the `read_mail` command you can also answer the message, save it in one (or more) of several kinds of segments, and forward copies to other users. As outgoing mail, after you create the message with the `send_mail` command you can edit both the text and the header information, save a copy for yourself, send it to one or many people, and receive an automatic acknowledgement as soon as those users read it.

#### REQUESTS (read mail AND send mail)

All of the `read_mail` and `send_mail` options are available by issuing requests in the command's request loop, a part of the mail system that reads the request you type, performs the specified operation, and finishes with a prompt to you for another request.

Request usage is governed by regular command language rules; therefore, you construct request lines just the way you construct command lines. For example, you can use semicolons to separate multiple requests on one line:

```
send_mail: ! print;send;quit
```

and parentheses can be used for iteration (repetition):

```
read_mail: ! (print delete) 1
```

Refer to The New Users' Intro for a review of the Multics command language.

## Control Arguments and Requests

Control arguments and requests in the mail system can occasionally become bewildering. The `read_mail` and the `send_mail` commands together have over 60 control arguments. Mail system requests often have the same names as control arguments. In addition, many requests have their own control arguments, some of which are identical to command control arguments! It is important to employ these terms at their correct level (command level or request level).

As noted in The New Users' Intro - Part I, a command typed to Multics, possibly including one or more control arguments, constitutes a command line:

```
! rdm -log -list
```

A request plus any of its arguments, called a request line, is typed after a mail system prompt:

```
read_mail: ! list OR send_mail: ! log
```

Be careful not to type a request on a command line:

```
! rdm print
read_mail: Entry not found. >udd>ProjCat>Willow>print.mbx
r 09:36 0.231 53
```

or a command control argument as a request line:

```
read_mail: ! -list
read_mail: Unknown request "-list". Type "?" for
a request list.
```

Control arguments for both command lines and request lines are discussed in this manual. For the sake of clarity, most references to control arguments explicitly indicate "command control argument" or "request control argument", in order to differentiate between the two levels. In examples, command lines always begin with the command's short name (`rdm`), and request lines with the prompt (`read_mail:`).

## HOW TO USE YOUR MAILBOX

A few pointers will help you to use your mailbox and the mail system successfully and effectively.

Keep your personal mailbox empty, either by reading and deleting its contents regularly, or by storing your messages elsewhere for later examination (see Section 6, "Storing Your Mail", for various ways to do this). This practice helps keep to a minimum the amount of mail you must read through each time you look at your mailbox.

Interactive messages are one-line messages sent, via the `send_message (sm)` command, directly to the recipient's terminal. The notice telling you that a mail system message has arrived is an interactive message:

```
From Moch.ProjCat 08/01/80 09:14 mst Fri: You have mail.
```

You cannot receive interactive messages such as this one until you issue the `accept_messages (am)` command. By far the easiest way to issue this command is to place it in your `start_up exec_com` segment, so that you accept messages automatically each time you log in. See the New Users' Intro - Part II for information about `exec_coms`, the `start_up.ec`, and accepting and sending interactive messages.

Another useful command to place right at the end of the `start_up.ec` is `read_mail (or print_mail)`, or the command line `rdm -list`. In this way you can check the contents of your mailbox immediately after you log in.

To learn more about including the mail commands in your `start_up.ec`, see Section 7 of this manual, "Advanced Mail Features".

## SECTION 2

### THE PRINT\_MAIL COMMAND

The `print_mail` command is a simple interactive command, designed for people who will be using the mail system infrequently.

Type the command name `print_mail` (short name `prm`). The command prints a banner telling you how many messages you have. If you have no messages, you are informed of this and returned to command level. If you have any messages, the command immediately prints your first message: header and then text. It also prints a line of information just before the header, noting who mailed the message and how many lines of text it contains.

After each message you are prompted for a response with the question "Delete #N?". For example:

```
! prm
  You have one message.

#1 (4 lines) 08/01/80 09:14 Mailed by: Moch.ProjCat
Date: 1 August 1980 09:14 mst
From: Moch.ProjCat
Subject: picnic
To: Willow.ProjCat

There will be a meeting at 9:30 on Tuesday to discuss
plans for the umpteenth annual office picnic.
Everyone is asked to attend--please inform
the others in your project.

print_mail: Delete #1? ! <type response here>
```



Six responses are available:

- ?  
print the list of acceptable responses, and then repeat the query
- yes (y)  
delete the message and go on
- no (n)  
do not delete the message, and go on
- reprint (print, pr, p)  
print the message again
- abort  
delete nothing and return to command level
- quit (q)  
delete as directed and return to command level

As soon as you type a response, another message is printed (unless you have typed ?, abort, or quit); if you have no further messages, a ready message is printed, indicating that you have returned to command level.

If you are in the middle of a long message and you decide you don't want to read any more, press the BREAK or QUIT key on your terminal. (See the New Users' Intro manual for a description of issuing the QUIT signal in this manner.) When the system responds with a QUIT message, type the program\_interrupt (pi) command, which returns you to the print\_mail query. You can then delete or save the message, and continue to the next message.

If you supply an incorrect response (for instance, if you misspell the response), the command suggests that you type a "?" for the list of responses. If you delete a message and then decide you still want it, use the abort response to return to command level, rather than the quit response; the abort response leaves your mailbox just the way it was when you issued the print\_mail command.

A useful control argument to the print\_mail command is -list (-ls). It prints a summary of your messages before going on to print the first message. Here is a sample for the above message:

```
! prm -ls
  You have one message.

Msg#  Lines  Date   Time   From           Subject:
  1*   (4) 08/01/80 09:14 Moch.ProjCat   picnic

<message #1 is printed here>
```

This control argument can refresh your memory and save you time, especially when used in conjunction with the QUIT signal. \*

## SECTION 3

### THE SEND\_MAIL COMMAND

The `send_mail` command provides you with the ability to create and send messages. It also gives you the opportunity to examine and edit your message before sending it, if you wish.

The first part of this section presents a review of the most basic use of the `send_mail` command. After reading this part, you can go directly to a terminal, write and deliver a message to another user, and be returned to command level. When you wish to learn more about the basic `send_mail` vocabulary of viewing, editing, sending, and gaining assistance, you can read on in this section. Later sections (5, 6, and 7) describe additional capabilities of the Multics mail system.

#### BASIC send mail COMMAND

Enter `send_mail` by typing the `send_mail` command (short name `sdm`) and the `User_id` of the person to whom you are writing. (Within the mail system, the `User_id` is considered one form of address, because the mail system uses this information to deliver the message to the correct mailbox.) Remember that a `User_id` consists of both a `Person_id` and a `Project_id`. After you type a newline, `send_mail` prompts you for the subject of your message:

```
! sdm Willow.ProjCat
  Subject:
```

(A subject line gives the recipient a very useful way of remembering what the message concerns.) Type in a title and another newline directly after this prompt. Now `send_mail` responds with another prompt, indicating that you may proceed with your message.

```
! sdm Willow.ProjCat
  Subject: ! and you?
  Message:
```

As you type in your message, keep in mind that the # and @ characters are always available for correcting or erasing the line you are currently working on.

The simplest way to conclude your message is to type a period alone on a line, and then a newline. As soon as you do this, the message is sent to the person you specified, and you receive a confirming system note that looks like "Mail delivered to Willow.ProjCat". Then a ready message is printed, indicating that you have been returned to command level automatically.

Here is an example of one complete session in send mail. Note the use of the # character to correct a mistake in the message text.

```
! sdm Willow.ProjCat
  Subject: ! and you?
  Message:
! Are you going to the picnic meeting on Thu##uesday? I hope
! to go, but I don't know if
! it will be possible.
!
  Mail delivered to Willow.ProjCat.
  r 10:26 0.272 94
```

### THE REQUEST LOOP

The send\_mail command has several requests that are as useful to the new user as to more experienced users. As you see from the example above, however, you have had no opportunity to give send\_mail any requests -- you are automatically returned to command level when you finish typing in your message. In order to issue requests, you must enter the send\_mail request loop. The request loop, described in "Requests" in Section 1, is a repeating cycle consisting of a send\_mail prompt, your request, and a resulting send\_mail action, followed by another prompt.

Several ways of entering the send\_mail request loop are explained in this section. One method is to end your message with a "\q" instead of a period. You will be answered with the send\_mail prompt, indicating that you are in the send\_mail request loop:

```
! sdm Willow.ProjCat
  Subject:  ! and you?
  Message:
! Are you going to the picnic meeting on Tuesday?  I hope
! to go, but I don't know if
! it will be possible.
! \q

send_mail:
```

At this point, you are ready to type any request you wish.

Other methods for entering the request loop are described in "Editing Your Message" just below, and in "send\_mail Command Control Arguments" at the end of this section. For now, though, simply type "\q" as the last line of your message.

### VIEWING YOUR MESSAGE

#### The print Request

The send\_mail print (pr) request displays the message text, and is preceded by a shortened version of the message header. The example message from above is used for illustration:

```
send_mail:  ! pr

(2 lines in text):
Subject:  picnic
To:  Willow.ProjCat

Are you going to the picnic meeting on Tuesday?  I
hope to go, but I don't know if it will be possible.

send_mail:
```

Notice that the message text does not appear just the way you typed it in. See "Message Filling" later in this section for a complete explanation.

When you want to see the entire message, header and all, use the `-header (-he)` control argument with `print`:

```
send_mail: ! pr -he
```

To view only the text of your message, use the `-no_header (-nhe)` control argument:

```
send_mail: ! pr -nhe
```

### The print header Request

The `print_header (prhe)` request enables you to see the complete header of a message, without its text:

```
send_mail: ! prhe

(1 line in text):
Date: 1 August 1980 09:14 mst
From: Moch.ProjCat
Subject: picnic
To: Willow.ProjCat

send_mail:
```

To obtain just the shortened header, as illustrated for the `print` request above, add the `-brief (-bf)` control argument:

```
send_mail: ! prhe -bf
```

\*

### EDITING YOUR MESSAGE

One of the most useful aspects of `send_mail` is its built-in editor. A version of the `gedx` editor, it allows you to change, delete, and add to your message while you remain in `send_mail`. However, you do not need to type "w" before you end your editing session -- the editor does this automatically.

The `send_mail` editor operates like the `gedx` editor introduced in the `New Users' Intro - Part I`, and explained fully in the `Qedx Users' Guide`. You are strongly encouraged to turn to one or both of those manuals, because in this manual only a review of the simplest subset of editor requests is given.

When you are first typing in your message and you want to make changes, type "\f" alone on a line, just as you do in qedx when you wish to move from input mode to edit mode:

```

Message:
! There will be a meeting at 11:00#
! \f

```

When you are already in the send\_mail request loop and you want to enter the built-in editor, you should use the qedx (qx) request:

```
send_mail: ! qx
```

Once you are in the editor, you issue editor requests, as opposed to send\_mail requests. Here is a list of basic editor requests:

<u>REQUEST</u>	<u>DESCRIPTION</u>	<u>EXAMPLES</u>
p	prints the specified line(s)	p 2p 1,3p
=	prints the line number of the specified line	= \$=
d	deletes the specified line(s)	d 3d 1,\$d
a	adds lines of text after the specified line	a 2a
s/old/new/	substitutes every occurrence of the first character string with the second character string, on the specified line(s)	s/hte/the/ 1,\$s/11:00/9:30/
s/old/new/p	same as above, but also prints the changed line	s/hte/the/p

If you wish to abort all changes made within the editor, type the qedx request 1,\$dr on a line by itself. This restores the original message text to the qedx buffer.

To leave the send\_mail editor, simply type q and you will be returned to the send\_mail request loop. Note that this q request is the editor quit request, not the send\_mail quit request (see "Quitting" below).

Here is an extended example of how an answer to the previous message could be constructed. Supplemental comments are displayed to the right of the example. Spaces that would not necessarily be in an actual session are included here for clarity.

```

! sdm Willow.ProjCat
  Subject: ! your talk
  Message:
! I thik your talk           <a first draft>
! was good.
! If you wtanto to @       <a first draft>
! If you would like more spcefic
! comments, let me know.
! \f                       <enter edit mode>

! ls/k/nk/p                <correct one error, >
  I think your talnk      < but cause another!>

! s/lnk/lk this morning/   <fix second error on >
                           < current line, and >
                           <      add more info >

! s/ink/ought/            <another change>

! 1,$p                    <print entire message>
  I thought your talk this morning
  was good.
                           .
                           .
                           .
  If you would like more spcefic
  comments, let me know.   <--
! 4p
  If you would like more spcefic
! s/spce/speci/p          <correct another error,>
  If you would like more specific < and print the line >

! 3d                      <delete empty line>

! q                        <leave editor>

send_mail: ! send

Mail delivered to Willow.ProjCat.

send_mail: ! quit
r 13:02 0.478 92

```

Further editing features are discussed in Sections 5 ("Mail Segments") and 7 ("The apply Request").



## SENDING YOUR MESSAGE

Once you are in the `send_mail` request loop, it is important to know the `send` request -- otherwise your message will not get delivered. The `send_mail` command delivers mail automatically only when you bypass the request loop by ending your message with ".", as described in "Basic `send_mail`" at the beginning of this section.

The simplest way to use this request is to type `send`. If you entered the `send_mail` command with an address, as described in the beginning of this section, then the message is immediately sent to the mailbox of the person you specified on the command line. A notice is printed confirming delivery, as well as the usual `send_mail` prompt:

```
send_mail: ! send
Mail delivered to Willow.ProjCat.

send_mail:
```

If the message cannot be delivered, you receive immediate notice of the cause:

```
send_mail (send): Some directory in the path specified
does not exist. >udd>ProjCat>Willow>Willow.mbx
send_mail (send): The message was not sent.

send_mail:
```

The cause here was a missing "l" in the `User_id` (which you can correct with the `remove` and `to` requests, described in Section 5).

You may ascertain that the recipient of your message has read the message by supplying the `-acknowledge` (`-ack`) request control argument with the `send` request. When the person reads your message, you automatically receive an interactive message like this one:

```
From Willow.ProjCat 08/01/80 15:41 mst Fri:
Acknowledging your message of 1 August 1980 09:14 est;
Subject: your talk
```

Another consequence of using -acknowledge with the send request is that it adds an extra field to the message header:

```
(2 lines in text):  
Date: 1 August 1980 13:02 mst  
From: Merce.ProjDog  
Subject: your talk  
To: Willow.ProjCat  
Acknowledge-To: Merce.ProjDog
```

The acknowledgement is sent by the mail system from the recipient's mailbox automatically.

### MESSAGE FILLING

Once you send your message by typing "." alone on a line followed by a newline, the message is automatically reformatted. The right margin of the text is adjusted so that no line has more than a certain number of characters. This process of message reformatting is called FILLING. For example, when user Willow reads the text of the picnic message, it looks like this:

```
There will be a meeting at 9:30 on Tuesday to discuss  
plans for the umpteenth annual office picnic. Everyone  
is asked to attend -- please inform the others in your  
project.
```

If you type a message online and then use the qedx editor before sending it, the message is filled automatically after you exit qedx. See Appendix A for further details on filling in qedx within send\_mail.

Within send\_mail, the fill (fi) request allows you to fill the message text as described above, and to set the line length of the filled text. By default, the maximum line length of filled text is set at 72 characters. If you prefer, you can specify another length with the -line\_length (-ll) control argument followed by the maximum number of characters you want:

```
send_mail: ! fi -ll 50
```

This makes the message text look like this:

```
There will be a meeting at 9:30 on Tuesday to
discuss plans for the umpteenth annual office
picnic. Everyone is asked to attend -- please
inform the others in your project.
```

The `send_mail` control argument `-line_length` (`-ll`) also formats the text in the manner described above.

### QUITTING

Leaving `send_mail` is usually easy; just type "quit" or "q". When you have left unfinished business, though, `send_mail` checks to make sure that you really want to exit:

```
send_mail: ! q
send_mail (quit): Message has not been sent, saved, or
written. Do you still wish to quit?
```

If you purposely wish to leave `send_mail` without sending a message, you can avoid `send_mail`'s query with the `-force` (`-fc`) control argument to the quit request:

```
send_mail: ! q -fc
r 13:07 0.332 116
```

As the ready message shows, you are immediately returned to command level.

### ASSISTANCE

The `send_mail` command has four means of assistance available while you are working.

## The ? Request

When you forget the name of a request, or which letter is the short name for what request, type the ? request. It prints a multi-columnar list of all requests and their short names. Here is an abbreviated version of the ? request and response, listing only the requests discussed in this section:

```
send_mail: ! ?
```

```
Summary of send_requests:
```

```
quit, q      print, pr, p    fill, fi      help
send         qedx, qx      print_header, prhe
```

```
Type "list_requests" for a short description of the
requests.
```

```
send_mail:
```

## The list requests Request

If you want to obtain a brief description of the available requests, type the list\_requests (lr) request. It prints a list of all requests, plus a memory-jogging, one-line description of each request. The lr request also provides several lines of significant information preceding the list of requests. Here is an example of the list\_requests request and response (only the requests already discussed in this section are listed):

```
send_mail: ! lr
```

```
Summary of send_mail requests:
```

```
use ".. COMMAND_LINE" to escape a command line to Multics.
Type "list_help" for a list of topics available to
the help request.
Type "help TOPIC" for more information on a given topic.
```

```
quit, q      Leave send_mail.
print, pr, p Print the specified message.
print_header,
  prhe      Print the message's header.
qedx, qx     Edit the message.
fill, fi     Reformat text to fit given width.
help        Obtain detailed information on send_mail.
?           Produce a list of the most common requests.
```

```
send_mail:
```

In addition, you can specify a topic name with the lr request, and receive a list of all requests which contain that topic name. For example, you may want to know what requests contain the word "list" in send\_mail:

```
send_mail: ! lr list

list_help, lh      List topics for which help is available.
list_requests, lr List brief info on send_mail requests.

send_mail:
```

### The help Request

For detailed information on how to use a particular request, type "help" followed by the name of the request:

```
send_mail: ! help quit
(6 lines follow; 16 in info)
09/26/82 send_mail request: quit, q

Syntax: quit {-control_args}

Function: exits send_mail.

Control arguments (8 lines). More help? ! yes

Control arguments:
-force, -fc
  causes send_mail to exit even though the message has
  been modified since it was last sent, saved, or written.
-no_force, -nfc
  causes send_mail to query the user for permission to
  exit if the message has been modified since it was last
  sent, saved, or written. (Default)

send_mail:
```

The help request is similar to the Multics help command, but it is simpler and more restricted. It offers an internal set of info segments on every send\_mail request, and on selected other topics concerning send\_mail. For a list of topics, use the list\_help request (described below).

Most of the control arguments accepted by the Multics help command are accepted by the help request. The -brief (-bf) control argument is particularly useful; it produces a summary of the request, including the syntax line, arguments, and control arguments. For a complete description of the help request, type "help help".

The help request is a prompting request, asking you at several points if you want more information. The example above illustrates one of the possible responses to the help prompt: "yes". If you want a list of all the responses that you could give while inside the help request, type a "?" in answer to the help prompt.

If you type the help request with no arguments, you get a response which explains several ways to obtain online information.

### The list help Request

For a list of available info segments on send\_mail topics, type the list\_help (lh) request. If you specify a topic after the request, you receive a list of all send\_mail info segments pertaining to that topic. For example:

```
send_mail: ! list_help print
Topics available for send_mail:
print
print_header
send_mail:
```

### send mail CONTROL ARGUMENTS

All the control arguments discussed up to this point have been request control arguments, added to the request line after the request to which they belong -- as, for example:

```
send_mail: ! pr -nhe      or      send_mail: ! q -fc
```

The send\_mail command itself also has a set of control arguments, as noted in Section 1; you include them on the command line, after typing "send\_mail" and a User\_id. Here are two that may be useful to you.

The best method for entering the send\_mail request loop is via the command control argument -request\_loop (-rql):

```
! send_mail Willow.ProjCat -rql
```

After you are prompted for the subject and text of your message, you may conclude the text with a period, and you will be greeted with a send\_mail prompt:

```
! <text of message>
! .

send_mail:
```

An interesting and handy control argument is called -input\_file <path> or -if <path>. This permits you to send a regular ASCII segment as a message. For instance, a list of picnic foods in a segment named "victuals" can be sent this way:

```
! sdm Willow.ProjCat -if victuals
  Subject: ! picnic stuff

send_mail: ! send; q
Mail delivered to Willow.ProjCat.
r 16:11 0.291 86
```

The segment that you send should contain only the message text, because send\_mail supplies the message header.

Notice that when using the -input\_file control argument you can still provide a subject for the message. Also, you can use the built-in editor or other send\_mail requests, because you are put into the send\_mail request loop after you provide the message subject.

When sending a message using the -if control argument, the message text is not automatically filled. It is assumed that you have already formatted the file before sending it. If you wish to reformat the file while sending it, use the fill request. This request causes the text to be reformatted in the manner described in "Message Filling" earlier in this section. For example, user Moch.ProjCat sends an input file which is filled to the default line length of 72 characters, with the following lines:

```
sdm Willow.ProjCat -if victuals
Subject:    picnic stuff

send_mail:  fill; send
Mail delivered to Willow.ProjCat.

send_mail:
```

The -acknowledge (-ack) command control argument provides you with a confirmation of your message being read, without you having to enter the request loop.

The send\_mail command has many more control arguments. Most of them are presented in later sections of this manual. A complete list of the available control arguments is in Appendix A.



## SECTION 4

### THE READ\_MAIL COMMAND

The `read_mail` command is a flexible interactive command. It is designed to be completely accessible to the novice, and also useful for a variety of advanced purposes.

The first part of this section presents, in brief, the most basic use of the command. After reading this part, you can go directly to a terminal and perform the simplest tasks of reading and discarding one or more of your messages, and returning to command level. When you wish to learn more about the basic `read_mail` vocabulary, you can read on in this section. Later sections (5, 6, and 7) present additional capabilities within the `read_mail` and `send_mail` commands.

#### BASIC read mail REQUESTS

When you type `read_mail` (short name `rdm`), the command prints a banner telling you how many messages you have. It then skips a line, and types a prompt:

```
! rdm
  You have 2 messages.

  read_mail:
```

The command waits for you to type a read\_mail request in response to this prompt. (If you have no mail, a notice is printed telling you this, and you are returned to command level.) When you type a request, read\_mail performs the task you have requested and then prompts you again for another request. The four most basic requests are:

- list (ls) prints a heading line, and then one line of information about each message. The first column contains the message number, denoting the position of that message in the mailbox.

```
read_mail: ! ls

Msg#  Lines  Date      Time    From           Subject
 1*   (4) 08/01/80  09:14  Moch.ProjCat  picnic
 2    (2) 08/01/80  10:26  Brie.ProjDog  and you?

read_mail:
```

- print (pr, p) prints the header and text of the message or messages you specify; a message is specified by its message number. Type the message number directly after the request (e.g., print 1).

```
read_mail: ! pr 2

#2 (1 line) 08/01/80 10:26 Mailed by: Brie.ProjDog
Date: 1 August 1980 10:26 mst
From: Brie.ProjDog
Subject: and you?
To: Willow.ProjCat

Are you going to the picnic meeting on Tuesday? I
hope to go, but I don't know if it will be possible.
---(2)---

read_mail:
```

- delete (dl, d) deletes the message or messages you specify. Type the message number directly after the request.

```
read_mail: ! dl 2
read_mail:
```

- quit (q) returns you to command level.

```
read_mail: ! q
r 14:22 0.445 325
```

## LISTING AND PRINTING

### The list Request

The list (ls) request serves as a handy reference tool in many situations. It provides a one-line summary of relevant information about each of your messages; this aids in deciding what you want to do with them. Here is a sample list summary from a mailbox with four messages:

Msg#	Lines	Date	Time	From	Subject
1*	(4)	08/01/80	09:14	Moch.ProjCat	picnic
2	(2)	08/01/80	10:26	Brie.ProjCat	and you?
3	(2)	08/01/80	13:02	Merce.ProjDog	your talk
4	(27)	08/01/80	16:47	Edgar.ProjDog	comments y<MORE>

The Message Number column shows the position of each message in this mailbox at this time. The Lines column includes only the lines of text in a message, not the number of header lines. The date and time that the message was sent to you are recorded also, as is the person who sent it to you. If the sender has included a subject, the Subject column includes as much of the subject as will fit on the rest of the line. The asterisk next to a message number indicates which is the current message.

You can use the list request to give you a summary line about a single message; simply follow the request name with a message number:

```
read_mail: ! ls 4

Msg#  Lines  Date   Time   From           Subject
 4*   (27) 08/01/80 16:47 Edgar.ProjDog  comments y<MORE>
```

At the end of the summary line, "<MORE>" indicates that the title is longer than can fit on the line. Also notice the asterisk after message #4 -- listing a message makes it become the current message.

### The print Request

As noted above, the print (pr, p) request prints both header and text of the message or messages you specify. With a summary of messages in front of you, you can use the print request more effectively. If you have many messages, you can choose which message to print first, or you can decide not to read certain ones at this time.

### MESSAGE SPECIFIERS

In order to print your messages so far, you have issued the print request followed by a message number. A message number is one of several MESSAGE SPECIFIERS: ways of indicating which messages you want to see.

### Keywords

Another kind of message specifier is the keyword. These keywords are used just like message numbers:

- current (short name c)
- next (n)
- previous (p)
- first (f)
- last (l)
- all (a)

When you type "current" directly after the print request ("pr current"), you get the message that is currently being worked on by the read\_mail command. The current message is always message #1 at first, and it shifts when you issue a request that deals with some other message; for example, when you first enter read\_mail, message #1 is the current message, but when you type "print 2" then message #2 becomes the current one. You can also type simply "print" to see the current message.

The "next" and "previous" keywords refer to the messages relative to the current message, so they shift as the current message shifts. The "first" and "last" keywords operate on the first and last remaining messages in the mailbox.

### Ranges

There are also several ways to print more than one message at a time. When you know exactly which messages you want to see, you may type several message numbers separated by spaces:

```
! p 3 1 4
```

The messages are printed in the order you specify.

If you want to see several messages in a row, you can specify a range by typing a message specifier for the earliest message you want, then a colon, and then a message specifier (no intervening spaces) for the last message you want, like this:

```
! pr 2:4
```

This prints messages #2, #3, and #4 for you. The keyword "all" prints all the undeleted messages in your mailbox.

When specifying a range, you can use any combination of the above-mentioned message specifier types. For example, assuming there are four messages in your mailbox and message #1 is the current message, all of the following expressions yield the same result:

print f:last	p 1:4
pr c:4	pr 1 2 3 4
p all	pr c:last
print 1:3 last	

For further information on message specifiers, see Appendix A.

## print REQUEST CONTROL ARGUMENTS

In some cases you know that you will not want to keep a particular message after you read it. The `-delete (-dl)` control argument is useful then:

```
read_mail: ! p first -dl
```

This request line is equivalent to:

```
read_mail: ! p first;d first
```

After the message you specify is printed out for you, it is deleted.

If you wish to bypass printing the full header when reading a message, you can supply the print request with its `-no_header (-nhe)` control argument. A shortened header is then printed before the text of the message, including only essential information:

```
read_mail: ! pr 3 -nhe
```

```
#3 (2 lines) 08/01/80 13:02 Mailed by: Merce.ProjDog  
From: Merce.ProjDog  
Subject: your talk
```

```
I thought your talk this morning was good. If you  
would like more specific comments, let me know.
```

```
---(3)---
```

```
read_mail:
```

The first line of the shortened header includes the date and time the message was sent to this mailbox, which can be different from when the message was written or first sent.

There may be times when you need more information about a message than you can get from the list request, but you don't want to read through the text of the message. The `read_mail` request `print_header` functions just as the `send_mail print_header` request does:

```
read_mail: ! prhe 3

#3 (2 lines) 08/01/80 13:02 Mailed by: Merce.ProjDog
Date: 1 August 1980 13:02 mst
From: Merce.ProjDog
Subject: your talk
To: Willow.ProjCat

read_mail:
```

The need for the `print_header` request occurs more frequently as you (or the people sending you messages) learn to send messages in more complex ways. Several of the additional `read_mail` and `send_mail` requests add extra header fields to message headers.

#### REPLYING TO MESSAGES

In many cases the most efficient way of responding to the messages you receive is with the `reply (rp)` request. When you supply the `reply` request with one message specifier, you are immediately placed in `send_mail` and prompted for the text of your reply:

```
read_mail: ! rp 2
Replying to Brie.ProjDog
Message:
```

The subject of your message is automatically taken from the Subject field of the message you are replying to:

Subject: Re: and you?

unless you specify another subject with the `send_mail subject` request. You can reply to only one message at a time.

To send the reply, simply type a period, as you would a regular message. Because you create the reply using `send_mail`, you can also type `"\q"` to enter the `send_mail` request loop. When you leave `send_mail` (via the quit request or `."`), you are returned to `read_mail`.

When reply is used, the In-Reply-To field is added to the message header of the reply:

| In-Reply-To: Message of 1 August 1980 10:26 mst from Brie.ProjDog

This tells the recipients which message is being answered. Many of the send\_mail command control arguments can be used on the reply request line. See Appendix A for details on this request.

#### FORWARDING A MESSAGE

| You have the option of sending on copies of the messages you receive, with the forward (fwd, for) request. Follow the request name with the message specifier and the address(es) of recipients:

read\_mail: ! fwd 1 Scout.ProjCat When you use forward, several new fields are added to the message header:

```
Redistributed-Date: 1 August 1980 15:32 mst
Redistributed-By: Willow.ProjCat
Redistributed-To: Scout.ProjCat
```

This indicates to recipients how the forwarding was performed.

#### DELETION AND RETRIEVAL

##### The delete Request

| Once you have read a message and kept it in this mailbox as long as you want, you can delete it from your mailbox easily with the delete (dl, d) request and a message specifier. In fact, you may include several message specifiers in your delete request line:

```
read_mail: ! d 4 2
read_mail:
```



Notice that message specifiers may appear in any order, and they may have any number of spaces separating them. When you issue the list request after deleting messages, you receive a summary of the remaining messages, still with their original message numbers:

```
read_mail: ! ls

Msg#  Lines  Date   Time   From           Subject:
1      (4)  08/01/80 09:14  Moch.ProjCat   picnic
3*     (2)  08/01/80 13:02  Merce.ProjDog  your talk

read_mail:
```

Message numbers do not get reassigned to the remaining messages until you quit the read\_mail command.

If you try to delete a message which hasn't been listed, printed, saved, or written, you are queried with a prompt:

```
read_mail: ! dl 3

read_mail (delete): Message #3 has not been processed.
OK to delete? ! no
read_mail (delete): No messages deleted.

read_mail:
```

If you answer "no" to the query, no messages are deleted, as in the example above. If you answer "yes", the message is deleted. There is no acknowledgment of the deletion; you are simply prompted for another request.

When you have deleted each message in the mailbox, you are sent the notice:

All messages have been deleted.

## The retrieve Request

Deleted messages are not really deleted. They are merely "marked for deletion". They actually remain in the mailbox until you leave the mail system (with the quit request) and return to command level. If you have not yet left read mail, you can return your deleted messages to your mailbox by issuing the retrieve (rt) request with the message numbers of your deleted messages:

```
read_mail: ! dl 4
read_mail: ! rt 2 4
read_mail:
```

Because message numbers are not reassigned when a message is deleted, you simply type the message number that that message had before you marked it for deletion. Other forms of message specifier should not be used.

To check on the correct message number of a deleted message, type the list request with the `-include_deleted (-idl)` control argument. Assume the current message is message #2, and observe the following:

```
read_mail: ! dl 2
read_mail: ! ls -idl

Msg#  Lines  Date   Time   From           Subject:
1      (4)   08/01/80 09:14  Moch.ProjCat   picnic
2!     (2)   08/01/80 10:26  Brie.ProjDog   and you?
3*     (2)   08/01/80 13:02  Merce.ProjDog  your talk
4      (27)  08/01/80 16:47  Edgar.ProjDog  comments y<more>

read_mail:
```

The `-include_deleted` control argument to the list request lists all messages, including deleted ones. An exclamation point beside a message number signifies a deleted message. Note that once message #2 is deleted, the current message automatically becomes #3.

The print request also has the -idl control argument, performing the parallel operation with deleted messages. If message #2 has been deleted, then this request line:

```
read_mail: ! p 1:3
```

prints only messages #1 and #3, but this line:

```
read_mail: ! p 1:3 -idl
```

prints messages #1, #2, and #3.

Remember: no message is truly gone until you issue the quit request. Once you leave read\_mail, though, you can no longer retrieve deleted messages.

### QUITTING

All you need to do to leave read\_mail is type quit, or just q. But even the quit request has a couple of special features.

If you have been trying out various combinations of lists, message specifiers, deleting, and retrieving, you may be confused and worried about quitting and possibly deleting messages that you want to keep. Now is the time to use the -no\_delete (-ndl) control argument of the quit request:

```
<too many requests>
```

```
read_mail: ! q -ndl  
r 11:43 0.343 133
```

This discards all modifications that you have made during this session with read\_mail. Next time you enter read\_mail you will find your mailbox just the way you found it this time (plus any messages that have arrived since then). This control argument can be better than aspirin.

Sometimes when you issue the quit request you receive a note like this:

```
read_mail (quit): A new message has arrived. Do you
still wish to quit?
```

You must answer either yes, in which case you are returned to command level, or no, which gives you another read\_mail prompt. If you use the -force (-fc) request control argument with quit:

```
read_mail: ! q -fc
r 11:43 0.0703 286
```

you are returned to command level with no questions asked.

### ASSISTANCE

The read\_mail command has several means of assistance available while you are working.

### The ? Request

When you forget the name of a request, or which letter is the short name for what request, type the ? request. It prints a multi-columnar list of all requests and their short names. Here is an abbreviated version of the ? request and response, listing only the requests discussed so far in this section:

```
read_mail: ! ?
```

```
Summary of read_mail requests:
```

```
help      print, pr, p    delete, dl, d    reply, rp
quit, q    list, ls        retrieve, rt
```

```
Type "list_requests" for a short description of the
requests.
```

```
read_mail:
```

## The list requests Request

If you want to obtain a brief description of the available requests, type the `list_requests (lr)` request. It prints a list of all requests, plus a memory-jogging, one-line description of each request. The `lr` request also provides several lines of significant information preceding the list of requests. Here is an example of the `list_requests` request and response (only a few of the requests already discussed in this section are listed):

```
read_mail: ! lr
Summary of read_mail requests:

use ".. COMMAND_LINE" to escape a command line to Multics.
Type "list_help" for a list of topics available to
the help request.
Type "help TOPIC" for more information on a given topic.

quit, q           Leave read_mail.
print, pr, p      Print the specified messages.
list, ls          List the specified messages.
delete, dl, d     Delete the specified messages.

read_mail:
```

In addition, you can specify a topic name with the `lr` request, and receive a list of all requests which contain that topic name. For example, you may want to know what requests contain the word "list" in `read_mail`:

```
read_mail: ! lr list

list, ls          List the specified messages.
list_help, lh     List topics for which help is available.
list_requests, lr List brief info on read_mail requests.

read_mail:
```

## The help Request

For detailed information on how to use a particular request, type "help" followed by the name of the request:

```
read_mail: ! help quit
(6 lines follow; 27 in info)
09/28/82 read_mail request: quit, q

Syntax: quit {-control_args}

Function: deletes any message marked for deletion and
         exits read_mail.

Control arguments (7 lines). More help? ! yes

Control arguments:
-delete, -dl
    specifies that messages marked for deletion should indeed
    be deleted before exiting. (Default)
-no_delete, -ndl
    specifies that messages marked for deletion are not to be
    deleted.

7 more lines. More help? ! no

read_mail:
```

The help request is similar to the Multics help command. It offers an internal set of info segments on every read\_mail request, and on selected other topics concerning read\_mail. For a list of the topics, use the list\_help request described below.

The help request is a prompting request, asking you at several points if you want more information. The example above illustrates two of the possible responses to the help prompt, "yes" and "no". If you want a list of all the responses that you could give while inside the help request, type a ? in answer to the help prompt.

Most of the control arguments accepted by the Multics help command are accepted by the help request. The -brief (-bf) control argument is particularly useful; it gives you a summary of the request, including the syntax line, arguments, and control arguments. For a complete description of the help request, type "help help".

If you type the help request with no arguments, you get a response which explains several ways to obtain online information.

### The list help Request

For a list of available info segments on read\_mail topics, type the list\_help (lh) request. If you specify a topic after the request, you receive a list of all read\_mail info segments pertaining to that topic. For example:

```
read_mail: ! list_help print

Topics available for read_mail:

print
print_header

read_mail:
```

### read\_mail CONTROL ARGUMENTS

All the control arguments discussed up to this point have been request control arguments, added to the request line after the request to which they belong -- as, for example:

```
read_mail: ! pr 4 -nhe or read_mail: ! q -fc
```

The read\_mail command itself also has a set of control arguments; you include them on the command line, just after typing "read\_mail".

One command control argument may be of particular use to you at this point. By now you may rely on the list request so much that you would like to see a list of your messages as soon as you enter `read_mail`. In this case, use the `-list (-ls)` control argument:

```
! rdm -ls
You have 4 messages.

Msg#  Lines  Date   Time   From           Subject
1*    (4)  08/01/80 09:14  Moch.ProjCat   picnic
2     (1)  08/01/80 10:26  Brie.ProjDog   and you?
3     (2)  08/01/80 13:02  Merce.ProjDog  your talk
4     (27) 08/01/80 16:47  Edgar.ProjDog  comments y<MORE>

read_mail:
```

After the list summary is printed, you are prompted for your first request.

You may wish to have your messages printed with the brief type of header each time you issue the print request, rather than seeing the complete header. To have this as your default action, add the `-no_header (-nhe)` command control argument to the `read_mail` command line:

```
! rdm -nhe
```

For those times that you do wish to see the full header, you can specify the `-header (-he)` request control argument on the print request line:

```
! read_mail: p -he
```

The `read_mail` command has many more control arguments. Most of them are presented in later sections of this manual. A complete list of the available control arguments is in Appendix A.



## SECTION 5

### MORE ON SENDING A MESSAGE

With the `send_mail` command, you have learned how to create, edit, and send a message to one person. The first part of this section describes various ways of sending a message to as many users as you like.

Most of the requests described below affect the message header, because message headers contain the entire "history" of their messages, including information such as who sent the message and all the people who received it. So far, when you have sent messages, the mail system has gathered this information and automatically compiled the full header, with you adding only the title. In the second part of this section, you learn ways of modifying the header yourself.

#### SENDING TO SEVERAL PEOPLE

##### The to Request

The best way to send your message to several people is to use the `to request` in conjunction with the `send` request. There are many times when you already know all the people who should read a particular message. Perhaps it is also desirable that the recipients know who else receives the message. The `to request` lets you create a list of recipients for the message, which you can add to at any point:

```
send_mail:      to Edgar.ProjDog
```

When your message is completely ready to go, you just type the send request with no addresses, and the message gets delivered to all the people you've listed:

```
send_mail:    to Edgar.ProjDog
send_mail:    <other requests>
send_mail:    to FNewton.ProjDog
send_mail:    send
Mail delivered to Willow.ProjCat.
Mail delivered to Edgar.ProjDog.
Mail delivered to FNewton.ProjDog.

send_mail:
```

You may also type "to" without any addresses, to obtain the complete list of recipients:

```
send_mail:    to
To: Willow.ProjCat, Edgar.ProjDog, FNewton.ProjDog

send_mail:
```

Now if you type the print\_header request you will see an expanded To field in the message header:

```
send_mail:    prhe

(4 lines in text):
Date: 1 August 1980 09:14 mst
From: Moch.ProjCat
Subject: picnic
To: Willow.ProjCat, Edgar.ProjDog, FNewton.ProjDog
```

## The send Request

The most obvious way to send one message to several people is to use the send request several times:

```
send_mail:    send Edgar.ProjDog
Mail delivered to Edgar.ProjDog.

send_mail:    send FNewton.ProjDog
Mail delivered to FNewton.ProjDog.

send_mail:
```

This certainly works, and if you keep remembering more people to send the message to after you've already sent it, this is the quickest way. However, the fact that this message has been sent to two people does not appear in anyone's message header. You are the only person who knows all the people who received this message, when you use the send request.

Most requests that accept address arguments at all accept as many addresses as you want to type. A more efficient way of sending a message to the users listed above is:

```
send_mail:    send Edgar.ProjDog FNewton.ProjDog
Mail delivered to Edgar.ProjDog.
Mail delivered to FNewton.ProjDog.

send_mail:
```

In this situation, the default is that if the message cannot be delivered to one of the specified recipients (because of a misspelled address, for instance), it is not sent to any recipients. To reverse this default action, type the `-no_abort` control argument to the send request; now the message will be sent to all valid addresses.

Of course, you can accomplish the same result as above by typing the names of all the recipients on the `send_mail` command line:

```
send_mail Edgar.ProjDog FNewton.ProjDog
```

## The cc Request

You also have the option of sending "carbon copies" of a message to users who are not directly involved in the topic you are writing about, but who nevertheless are interested in or otherwise connected with the topic. The cc request thus simulates letter and memo procedure in a typical office environment.

Use this request just like the to request:

```
send_mail:    cc Scout.ProjCat Merce.ProjDog

send_mail:    cc
cc: Scout.ProjCat, Merce.ProjDog

send_mail:
```

You can also type the request without addresses, to see whom you already have on your cc list.

These secondary recipients will receive the message as soon as you type the next send request with no addresses:

```
send_mail:    cc Scout.ProjCat Merce.ProjDog

send_mail:    send
Mail delivered to Scout.ProjCat.
Mail delivered to Merce.ProjDog.
Mail delivered to Edgar.ProjDog.
Mail delivered to FNewton.ProjDog.

send_mail:
```

As the example shows, all recipients from all lists receive the message when you type the send request with no addresses, even if they have already received a copy. Thus, when using the to and cc requests, you should not issue a send request until after you have included all recipients.

When you do type send with addresses, only the people who are listed on this send request line receive the message at this time, even if you also have unprocessed lists of other recipients.

To see how the cc request changes a header, type the print\_header request:

```
send_mail:    prhe

(4 lines in text):
Date: 1 August 1980 09:14 mst
From: Moch.ProjCat
Subject: picnic
To: Willow.ProjCat, Edgar.ProjDog, FNewton.ProjDog
cc: Scout.ProjCat, Merce.ProjDog

send_mail:
```

The cc field has been added to the header information.

#### RELATED HEADER MODIFICATIONS

Once you are using the mail system for much of your written communication, you will probably start wishing that you could make more changes, not just in the text of your messages, but in the headers. What if you accidentally include an inappropriate address in the To field? How can you give an edited version of one message to a completely new set of people? Below are several more requests that help you tailor one message to suit varying requirements.

#### The remove Request

Almost as important as knowing how to add recipients for a message is knowing how to delete a recipient's name, before you send the message. This is one of several tasks that the remove request can easily accomplish for you.

The simplest way to delete addresses from lists of recipients is to type the remove (rm) request followed by all the addresses of those people whom you do not want to receive the message:

```
send_mail:    rm FNewton.ProjDog Scout.ProjCat
```

This request deletes FNewton and Scout from all lists in which they appear (if you had placed FNewton.ProjDog on both the To and the cc lists by mistake, both occurrences of that User\_id would now be deleted). The remove request control argument -all (-a):

```
send_mail:    rm -all
```

removes all addresses from the To and cc lists.

To be more specific as to which field you wish to delete from, you can use one of the remove control arguments. They are named after header fields, although the control arguments are, of course, in lowercase. For instance, to delete an address from only the cc field, this is the correct request line to type:

```
send_mail:      rm -cc Scout.ProjCat
```

Another remove control argument allows you to delete all addresses from the given field; use the -all (-a) control argument after the appropriate field control argument:

```
send_mail:      rm -cc -all
```

This also removes the cc field from the header.

If you become confused at any time about what you have done, remember that you can check the contents of any list by typing just the original request with no addresses (to or cc) or you can examine the entire header with the print\_header request.

### The subject Request

The title of a message, in the Subject field, may be both viewed and changed with the subject (sj) request:

<pre>send_mail:      subject                &lt;viewing the title&gt; Subject: picnic</pre>
<pre>send_mail:      sj meeting for picnic  &lt;changing the title&gt;</pre>
<pre>send_mail:</pre>

Following the subject request with a new title automatically deletes the previous title. In order to entirely erase the Subject field, use the remove request with the -subject (-sj) control argument:

```
send_mail:      rm -sj
```

In general, though, people appreciate seeing the subject of the messages they receive.

### The from Request

Although with you as the sender of a message, your User\_id must be present somewhere in the header, you may modify what is in the From field, with (what else?) the from request. This request is also useful for including several names or User\_ids:

```
send_mail:      from Willow.ProjCat Scout.ProjCat
```

when more than one person is responsible for the message.

When you change the From field, a new field is automatically added to the header -- the Sender field -- so as to indicate who actually delivered the message to this mailbox:

```
(1 line in text):  
Date: 1 August 1980 17:11 mst  
From: Willow.ProjCat, Scout.ProjCat  
Subject: that meeting  
Sender: Willow.ProjCat  
To: Moch.ProjCat
```

As with the other requests, issuing the from request alone gives you a look at what is currently in the From field. To remove the entry, use the "remove -from -all" request line; in this case, your User\_id replaces the previous entries, and the Sender field is deleted from the header.

### The -comment Control Argument

You can add information to the recipients' addresses in much the same way that you use the from request to supplement your own User\_id. The send\_mail command and any request that accepts addresses also accepts the -comment (-com) control argument, followed by a quoted character string. Here are two examples:

```
sdm Moch.ProjCat -comment FYI  
send_mail:      from Willow.ProjCat -com "Pil Willow"
```

Quotation marks are unnecessary for comments without blanks or punctuation marks, as in the first example above.

The resulting comment is placed in parentheses within the header, next to the address:

```
(3 lines in text):  
Date: 1 August 1980 16:21 mst  
From: Willow.ProjCat (Pil Willow)  
Subject: I can't go  
Sender: Willow.ProjCat, Scout.ProjCat  
To: Moch.ProjCat (FYI)
```

If you delete the commented address, any following comments are also deleted.



## SECTION 6

### STORING YOUR MAIL

The `read_mail` and `send_mail` commands have in common a group of requests that can store your mail in several types of segment, depending on how you plan to use the messages. Although there are slight differences between the `read_mail` and `send_mail` versions of the requests discussed in this section, the functions are the same for both.

#### YOUR LOGBOX

Just as every Multics user creates a personal mailbox for collecting incoming messages, everyone can have a default logbox made in which to log or keep messages. With this extra mailbox you can more thoroughly examine messages at your convenience, and yet keep your regular mailbox clear for new messages.

The logbox operates just like your regular mailbox. When you log messages from your personal mailbox into your logbox, the complete header as well as the text of each message is logged, ready to be examined. The only differences are that the logbox has a different name, and your mail does not get delivered directly to the logbox -- in fact, no users other than you are allowed to place mail in your logbox unless you give them permission by changing the extended access modes (Appendix B).

#### The log Request

As soon as you first use the log request (in either `send_mail` or `read_mail`), you receive a system note letting you know your logbox is being created. Its pathname is:

```
>udd>Project_id>Person_id>Person_id.sv.mbx
```

Notice the suffix ".sv.mbx" as part of the logbox name. You will probably never need to use this pathname, though, because in both `read_mail` and `send_mail` the log request delivers messages to your logbox automatically.

FROM read\_mail

When you are in read\_mail, and you wish to place copies of some messages into your logbox, type the log request, and message specifiers to indicate which messages you wish to log:

```
read_mail: ! log 2 4
```

If you would like your logged messages to be deleted from your regular mailbox, you may use either the delete request or the -delete (-dl) request control argument of the log request:

```
read_mail: ! log 2 4; dl 2 4 OR read_mail: ! log 2 4 -dl
```

You may also log already deleted messages (as long as you have not exited from read\_mail since you deleted those messages!) by using the log -include\_deleted (-idl) request control argument. Assuming message #1 has been deleted, this request line:

```
| read_mail: ! log 1:4 -idl
```

logs all four of the indicated messages. You may include the -delete request control argument along with the -idl request control argument here:

```
| read_mail: ! log 1:4 -idl -dl
```

which deletes the remaining undeleted messages (messages #2, #3 and #4) within that range.

FROM send\_mail

Inside send\_mail, you may log a copy of the message you are creating simply by typing the log request:

```
send_mail: ! log
```

No message specifiers or control arguments are necessary here, because you have only one message in send\_mail at a time.

You can also direct the send\_mail command to log messages by adding the send\_mail -log control argument to the command line as you enter:

```
! sdm Willow.ProjCat -log
```

By including -log on the command line, you are also adding your own User\_id to the cc header field.

## Examining Your Logbox

Because your logbox is one form of mailbox, you use the `read_mail` command to examine its contents. To specify that you want to see the logbox, include the `-log` command control argument after the command name, with any other command control arguments you want:

```
! rdm -log -list
```

The `-log` control argument causes `read_mail` to read only the contents of the logbox. All `read_mail` requests and control arguments are available for your use.

## ADDITIONAL MAILBOXES

### The save Request

The save request enables you to "file" your messages by topic, anywhere that you have access. This request creates extra mailboxes whenever and wherever you need them, and then, like the log request, stores the specified messages in whichever mailbox you indicate. This kind of mailbox is called a savebox; its pathname is:

```
>udd>Working_Directory>Name.sv.mbx
```

where "Working\_Directory" is the directory you are currently in, and "Name" is chosen by you. User Willow's "outgoing" savebox, in her "canine" directory, has this pathname:

```
>udd>ProjCat>Willow>canine>outgoing.sv.mbx
```

### WITHIN send\_mail

To create your first savebox for a message you are sending, type a save request line as if the desired savebox already existed. Following the `send_mail` prompt, type "save" and the pathname you have chosen for the new mailbox:

```
send_mail: ! save picnic_info
```

The mail system automatically adds on the ".sv.mbx" suffix and then verifies your intentions with this message:

```
send_mail (send): >udd>ProjCat>Willow>picnic_info.sv.mbx not found. Do you wish to create it?
```

Answer "yes", and a copy of your message is now stored in your new "picnic\_info" savebox.

If you know before you even enter `send_mail` that you will want to save the message you are about to create, you can enter `send_mail` with the `-save <path>` (`-sv <path>`) control argument to direct the coming message to the named savebox:

```
! sdm FNewton.ProjDog -save picnic_info
```

As with the `send_mail -log` control argument, your `User_id` is placed in the `cc_header` field, and a copy is saved in the specified savebox whenever the message is sent.

#### WITHIN `read_mail`

When you are in `read_mail`, you receive the same response as in `send_mail` from the mail system, when you use the save request with a new savebox name. In `read_mail`, though, you should supply message specifiers before typing the savebox name, to make clear which message or messages you wish saved:

```
read_mail: ! save 2 5 picnic_info
```

The `read_mail save` request allows the use of a `-delete (-dl)` request control argument:

```
read_mail: ! save 2 5 picnic_info -dl
```

so that you can clear your regular mailbox of messages as soon as they have been placed elsewhere. It also allows the `-include deleted (-idl)` request control argument, described above in "The Log Request".

#### The send Request

Among its many other features, the `send` request includes the two control arguments `-log` and `-save <path>`. These request control arguments perform the same actions to the messages specified on the `send` request line as do their request namesakes. For example, this request line:

```
send_mail: ! send Scout.ProjCat -save picnic_info
```

sends the message to `Scout.ProjCat` and saves a copy in the sender's `picnic_info.sv.mbx` savebox, just as a separate `save` request would do.

## EXAMINING OTHER MAILBOXES

### Your Saveboxes

You can examine one of your saveboxes in a `read_mail` command line, giving the name of the savebox as the argument:

```
rdm picnic_info
```

This places you inside your `picnic_info.sv.mbx` savebox, ready to read the messages you have stored here. The ".sv.mbx" suffix is added automatically.

There is one exception to the above method of examining saveboxes. If you type this line and you have a mailbox with the same name (i.e., `picnic_info.mbx`), you will be placed inside your `picnic_info.mbx` mailbox. To avoid this kind of confusion, give your saveboxes and mailboxes different names. However, if you have a savebox and a mailbox with the same name, you can enter you savebox in the following way:

```
rdm picnic_info.sv.mbx
```

Another way to look at one of your saveboxes is to use the `-save <path>` (`-sv <path>`) control argument in a `read_mail` command line, giving the name of the desired savebox as the `<path>`:

```
rdm -save picnic_info
```

### Other People's Mailboxes

You also have access to read and delete any messages that you have sent to other users' mailboxes. To do this, issue the `read_mail` command with an address on the command line. The address can be a `User_id` or the pathname of the mailbox you wish to examine:

```
rdm Moch.Projdog OR rdm >udd>ProjDog>Moch>Moch.mbx
```

Sometimes an address can be ambiguous; if this is the case, you can clarify the address by using one of two address control arguments, `-user <User_id>` or `-mailbox <path>` (`-mbx <path>`) like this:

```
prm -user Moch.BCD OR prm -mbx >udd>BCD>Moch>Moch.mbx
```

The ".mbx" suffix is assumed if you do not type it.

## MAIL SEGMENTS

Although the mail system itself offers you an impressive range of editing, storage, and distribution capabilities, you may find it very useful to be able to treat groups of messages as standard ASCII segments. You are then free to manipulate messages as you do other segments, to order printed copies, and to edit and add comments to any part of the message easily. When you use one of the requests described below to create a mail segment, standard access rules apply, because they are standard segments.

### The append Request

When in `read_mail`, place messages into a segment with the `append (app)` request, appropriate message specifiers, and the name of a segment:

```
read_mail:      append f:3 canine
```

Unless you have previously created it, this causes the segment "canine.mail" to be created in your directory, after an inquiry from `read_mail` to make sure this is what you had in mind. If the segment already exists, these messages are added to the end of the segment. In `read_mail` you can use the `-delete (-dl)` request control argument with `append` to delete the original message, as you can with the other requests discussed above.

In `send_mail`, simply type the request and pathname (the ".mail" suffix is added automatically):

```
send_mail:      append canine
```

The `send_mail` command also questions you about this segment if it has not yet been created.

Using one of these segments is just like using any regular segment -- but remember that pathnames of all mail segments end with the ".mail" suffix. When outside the mail system, be careful not to type "canine" when you mean "canine.mail".

### The write Request

The `write (w)` request is identical to the `append` request, with two additions. It has a `-truncate (-tc)` request control argument, which enables you to empty an existing mail segment of any previous contents before refilling it. There is also the `-extend` request control argument that adds to the existing segment, just as the `append` request does. This control argument represents the default action of the request.

### The preface Request

The preface (prf) request is very similar to the append | request, except that messages get added at the beginning of the segment specified, rather than at the end of the segment. This is useful for creating segments in which you want your newest messages to appear first.

## SECTION 7

### ADVANCED MAIL FEATURES

Many of the mail system components already discussed, such as control arguments on the command line, message editing, and powerful request language, also have additional features that enable you to use the `read_mail` and `send_mail` commands to meet more specialized requirements. These advanced techniques make use of command level capabilities from within the mail system.

#### ABBREVIATIONS

Within `read_mail` and `send_mail`, you can create abbrevs for request lines that you use frequently. These abbrevs can be expanded at your discretion. On the `read_mail` and `send_mail` command lines, the `-abbrev (-ab)` control argument turns on the abbrev process. In the request loop of `read_mail` and `send_mail`, the abbrev request acts in the same manner.

For example, you may forward mail frequently to `Merce.ProjDog`. Create the following abbrev at command level:

```
.a fwm forward c Merce.ProjDog
```

In `read_mail`, type the following to send the current message to `Merce.ProjDog`:

```
read_mail: ! abbrev
read_mail: ! fwm
Mail delivered to Merce.ProjDog
read_mail:
```



You can create an abbrev profile specifically for use within the mail system with the `-profile (-pf)` control argument. (A profile is a special segment in your `home_dir` containing your abbrevs.) This is helpful if, for example, you want to use the same short name "quit" for two different abbrevs: one within the mail subsystem, and one at command level. To specify the use of the profile `mail_system.profile`, type the following request line while in `read_mail`:

```
read_mail: ! abbrev -profile [e hd]>mail_system
```

You will now use `mail_system.profile` until you either quit `read_mail`, turn off the abbrev processor, or change to another profile. You can change profiles as often as you wish within the mail subsystem.

If you use a separate abbrev profile regularly, you may want to add the profile to your `read_mail` or `send_mail` abbrev. To use a special profile within `read_mail` automatically, create an abbrev similar to the following:

```
.ab Rdm do "read_mail -abbrev -profile [hd]>mail_system  
&rf1"
```

You can turn off the abbrev processor with the control argument `-no_abbrev (-nab)`. This control argument is the default. With it, you can override a command level abbrev for `read_mail` or `send_mail` that automatically turns on abbrev processing. For example, if you have the `Rdm` abbrev described above, you can enter `read_mail` and turn abbrev processing off like this:

```
Rdm -no_abbrev
```

Whenever conflicting control arguments appear on a command line; the mail system uses only the last one to appear. Thus the above example turns off abbrev processing as you enter `read_mail`.

Another way to turn off abbrevs within the mail subsystem is with the following request:

```
send_mail: ! abbrev -off
```

When creating abbrevs within read\_mail and send\_mail, a useful request is the do request. This request is identical to the do command, except that it executes a request line within read\_mail or send\_mail, rather than a Multics command line. Similarly, the if and answer requests are like the if and answer commands, but they operate within the context of a mail subsystem. The do and if commands are documented in the Intro to Multics - Part II manual. All three of these requests are useful in the creation of abbrevs within the mail system.

#### MORE ON CONTROL ARGUMENTS

Fourteen read\_mail and send\_mail command control arguments have been described in earlier sections of this manual. There remain approximately fifty others, nearly half of which represent actions that the command performs by default. The purpose of such an extensive set of controls is to let you create your own desired read\_mail and send\_mail environments. To illustrate a few simple possibilities, here are several example command lines, using some control arguments that you know and some that have not been discussed:

```
rdm -print -quit
rdm -list -no_header
```

The first read\_mail command line above merely prints any messages that you have and quits, returning you directly to command level. The second example prints a list of all messages in the mailbox before giving the read\_mail prompt; whenever you issue a print request, the message is printed with the brief form of header, just as if you had included the -no\_header print request control argument each time.

```
sdm Moch.ProjCat -acknowledge -save outgoing
sdm Moch.ProjCat -fm Willow.ProjCat -cmt "Pil Willow" -to
```

In the first send\_mail example, the message you create is acknowledged automatically when the recipient prints it, and a copy of your message will be saved in your mailbox "outgoing.sv.mbx". The second example places the comment Pil Willow after the User\_id Willow.ProjCat in the From field of the message header; the -to control argument is added so that all addresses typed afterward will be included in the To field.

## Control Arguments and start up.ec Segments

Another method of setting up your own read\_mail environment is to place a read\_mail command line, such as the one used in the previous example, at the end of your start\_up exec\_com segment. In this case, when your start\_up.ec has completed, you will be placed directly in read\_mail. If you have no mail, you receive a notice to that effect and are returned to command level. If you do have mail, you receive a list of your messages and then a read\_mail prompt. Here is an illustration:

```
Multics MR8.0: Honeywell LISD Phoenix, System M
Load = 102 out of 125.0 units: users = 109 08/01/80 ...
login Willow ProjCat
Password:

<login information>

You have four messages.

Msg#  Lines  Date  Time  From           Subject:
  1*   (4)  08/01/80 09:14 Moch.ProjCat   picnic
  2     (1)  08/01/80 10:26 Brie.ProjDog   and you?
  3     (2)  08/01/80 13:02 Merce.ProjDog  your talk
  4    (27)  08/01/80 16:47 Edgar.ProjDog  comments y<MORE>

read_mail:
```

You can use your start\_up.ec for other mail system functions, also. For instance, you could have your messages printed offline for you automatically each time you log in, by employing the -request (-rq) command control argument, and an enter\_output\_request command line. This control argument enables you to give one or more read\_mail requests after it (the list of requests must be quoted if there are any blanks); the specified requests are performed automatically, without entering the read\_mail request loop. Place these two lines in your start\_up.ec:

```
rdm -rq "write all my; delete all; quit"
eor my.mail
```

You are not placed in read\_mail as you would have been in the previous example, because with this line you included the quit request as part of the read\_mail command line.

## ESCAPING TO COMMAND LEVEL

There are several ways to use the Multics command environment while you remain inside the mail system. This ability can be handy for a variety of purposes.

### The .. Escape

To issue a Multics command within `read_mail` or `send_mail`, simply type two periods directly after the prompt, followed by the command line:

```
read_mail: ! .. who OR send_mail: ! .. who
```

When the command has finished, you receive another `read_mail` or `send_mail` prompt.

You can use this escape to check on which mail segments and mailboxes you have in your directory (`.. list`) and to attend to other Multics activities when they occur to you (`.. sm Brie.ProjDog Let's eat.`) without having to end your mail session prematurely.

You are free to use any command language conventions and facilities in the same way you do outside the mail system. Active functions (discussed in the New Users' Intro - Part II) are especially useful in providing the command language with extra flexibility. Here are two examples:

```
read_mail: .. sm [last_message_sender] Sure, I'm hungry too
send_mail: .. eor [home_dir]>canine.mail
```

Standard quoting and semicolon conventions also apply when using the `..` escape.

## Re-entering the Mail System

A very convenient feature of the `..` escape is that from either `read_mail` or `send_mail` you can re-enter `read_mail` to examine another mailbox, using the methods illustrated in "Examining Other Mailboxes" of Section 6. For example, you can check the contents of your logbox while in your default mailbox:

```
read_mail: ! .. rdm -log -list
There is one message in your logbox.

Msg#  Lines  Date    Time    From           Subject
  1    (4) 08/01/80  09:14  Moch.ProjCat   picnic

read_mail (2):
```

Notice that this `read_mail` prompt looks somewhat different from the usual one. The number in parentheses indicates the recursion level -- how many times you have entered `read_mail` within one mail session.

If you can't remember which mailbox you're in, use the `.` request. This request prints one line of information about that mailbox, including the pathname, as well as the state of the messages contained in the mailbox:

```
read_mail 8.3 (level 2): Message #1 of 1.
>udd>ProjCat>Moch>Moch.mbx
```

If the mailbox is one of your default mailboxes, you receive a note rather than an explicit pathname:

```
read_mail 8.3 (level 2): Message #1 of 1.
Reading your logbox.
```

You can also re-enter `send_mail` from either `read_mail` or `send_mail`, to send a message to one person while creating another message for someone else. The resulting `send_mail` prompts look like the `read_mail` prompt shown above:

```
send_mail (2):
```

The `.` request is also available here:

```
send_mail 6.0 3 lines (unprocessed); Subject: your talk
```

In `send_mail` the `.` request gives you information about the message you are creating.

## ACTIVE REQUESTS

Just as active functions increase flexibility within Multics command lines, active requests allow mail system request lines more flexibility. The four most useful ones are:

in send_mail:	in read_mail:
subject (sj)	mailbox (mbx)
execute (e)	execute (e)

Active requests are hereafter referred to by their short names in this section, to distinguish them from requests.

The sj active request returns the current subject of the message you are working on. Wherever you type [sj] on a request line, the mail system replaces that with the current contents of the Subject field. Two ways of using this active request are:

```
send_mail: ! subject [sj] and lunch
```

to add the words " and lunch" to the existing Subject field (this example also employs the subject request), and:

```
send_mail: ! append [sj]
```

which places the created message in a mail segment with the Subject field as its name. (This last is best done with a one-word subject -- otherwise you will have embedded blanks in the name, which would result in an invalid pathname.)

With the e active request, you can incorporate Multics active functions into mail system request lines, thereby increasing your options still more! The way to do this is to enclose the e active request, a space, and then an active function inside brackets. Below are a few simple examples:

```
read_mail: ! save 3 [e home_dir]>feline
```

for saving mail when you are not working in your home directory;

```
send_mail: ! to [e last_message_sender]
```

for sending mail to the user who last sent you an interactive message;

```
send_mail: ! send [e contents people_at_work]
```

to send mail to each person whose User\_id is included in the segment "people\_at\_work";

```
read_mail: ! append all [e date]
```

to place all your messages in a mail segment with today's date as its name.

The e active request and the mbx active request, which returns the pathname of the mailbox you are currently reading, are most commonly used with the execute request, described below.

## MORE REQUESTS

### The execute Request

The execute request (not to be confused with the e (execute) active request!) performs a function very similar to that of the .. escape, because it also passes the following line to command level to be acted on. Before the command line reaches command level, however, it passes through the request processor. This means that all special request line syntax, such as the active request brackets described above, gets processed first. The results are placed in the command line, and then the line gets processed as a command line. To illustrate with the mbx active request, which returns the pathname of the mailbox you are currently reading, you can type:

```
read_mail: execute mbx_list_acl [mbx]
```

to see the access control list for that mailbox (for descriptions of all mailbox access commands see Appendix B). After this request line goes through the request processor, it looks like the command line shown below, although you do not see this intermediate step:

```
mbx_list_acl >udd>ProjCat>Willow>Willow.sv.mbx
```

and it is processed just like any other command line.

Remember that the mbx active request is not a Multics active function. If you forget this, and try typing:

```
read_mail: ! .. mbx_list_acl [mbx]
```

you would receive the error message "Segment mbx not found".

You may also use the e active request within an execute request line, of course. For example, if you have created a mail segment like the one in the last example in the previous section, you could get a printout of it in one of two ways, while in read\_mail:

```
| read_mail: ! .. eor [date].mail
```

or else:

```
read_mail: ! execute eor [e date].mail
```

### The apply Request

If you prefer another Multics text editor to `gedx`, you may use the `apply (ap)` request to edit your message while in `send_mail`. Once in the `send_mail` request loop, type `apply` and the name of the editor you wish to use. For example, to use Emacs (on a video terminal), type:

```
send_mail: ! apply emacs
```

The screen will be cleared and replaced by the message within an Emacs buffer. When you are finished with your editing, you must write out the changes you have made, by typing `^X^S`. Then type `^X^C` as usual, and the familiar `send_mail` prompt will appear. (See the New User's Intro - Part I or the Emacs Text Editor Users' Guide (Order No. CH27) for information on Emacs.)

The `apply` request operates by appending the pathname of the temporary segment (created to hold your message before you send it) to the command line you provided - in the above case it was a command that invokes a text editor. Therefore the `apply` request also allows you to utilize your own `exec_coms`, and any compatible subsystems that may have been created at your Multics installation.

### The exec com Request

Within `read_mail` and `send_mail`, you can use the `exec_com (ec)` request to invoke an `ec`. The `ec` request works like the `exec_com` command documented in New Users' Intro - Part II, except that it makes use of `read_mail` or `send_mail` requests, rather than command level command sequences.

A `read_mail ec` segment must have the suffix "`rdmec`", and a `send_mail ec` must have the suffix "`sdmec`". An `ec` ending with any other suffix will not work in the mail system. These suffixes are used to avoid confusion with Multics command level `ecs`.

When you invoke an `ec` request within the mail system, your working directory is automatically searched, and then the following directory:

```
>udd>Project_id>Person_id>Person_id.mlsys
```



If the ec is not found in either of these directories, you will get an error message.

User Willow.ProjCat named the following simple ec "mo.rdmech", and put it in the >udd>ProjCat>Willow>Willow.mlsys directory:

```
&command_line off
ls -fm Moch.ProjCat
&command_line on
&quit
```

In read\_mail, user Willow types the ec request and gets an appropriate response:

read_mail		ec mo			
Msg#	Lines	Date	Time	From	Subject:
1*	(4)	08/01/80	09:14	Moch:ProjCat	picnic
5	(8)	08/03/80	11:23	Moch:ProjCat	my plans
8	(14)	08/05/80	12:36	Moch:ProjCat	meeting

APPENDIX A

MAIL SYSTEM COMMANDS

---

print\_mail (prm)

---

---

print\_mail (prm)

---

print\_mail (prm)

The print\_mail command prints all the messages in a mailbox, querying the user whether to delete each one.

### SYNTAX AS A COMMAND

prm {mbx\_specification} {-ca}

### ARGUMENTS

#### mbx\_specification

specifies the mailbox to be printed. If not specified, the user's default mailbox (>udd>Project>Person>Person.mbx) is assumed. The mailbox must be specified in one of the following forms:

#### -log

specifies the user's logbox and is equivalent to:

-mailbox >udd>Project\_id>Person\_id>Person\_id.sv.mbx

#### -mailbox PATH

#### -mbx PATH

specifies the pathname of a mailbox. The .mbx suffix is assumed if it is not present.

#### -save PATH

#### -sv PATH

specifies the pathname of a savebox. The .sv.mbx suffix is assumed.

#### -user Person\_id.Project\_id

specifies the given user's default mailbox. This control argument is equivalent to:

-mailbox >udd>Project\_id>Person\_id>Person\_id.sv.mbx

#### STR

is any non-control argument. First it is interpreted as -mailbox STR; if no mailbox is found, it is interpreted as -save STR; if no savebox is found, it is interpreted as -user STR.

---

print\_mail (prm)

---

---

print\_mail (prm)

---

### CONTROL ARGUMENTS

-acknowledge

-ack

acknowledges messages that request acknowledgement. This is the default.

-brief

-bf

shortens the print\_mail notice of the number of messages in the mailbox.

-header

-he

prints the complete message headers with the message text. This is the default.

-interactive\_messages

-im

operates on interactive messages from send\_message (when accept\_messages -hold is in effect) as well as mail messages from send\_mail. This is the default.

-list

-ls

prints a summary of the messages in the mailbox before printing the first message.

-long

-lg

prints the long form of the print\_mail message count notice. This is the default.

-no\_acknowledge

-nack

does not acknowledge any messages.

-no\_header

-nhe

prints a shortened form of the message header for each message.

-no\_interactive\_messages

-nim

operates on send\_mail messages only, not on interactive messages sent by send\_message.

---

print\_mail (prm)

---

---

print\_mail (prm)

---

-no\_list

-nls

does not prints a summary of the messages. This is the default.

-no\_reverse

-nrv

prints the messages in ascending numeric order. This is the default.

-own

prints only those messages that the user has sent to the mailbox.

-reverse

-rv

prints messages in reverse order.

#### QUERY RESPONSES

?

a list of the acceptable responses is printed, and the question is asked again.

yes

y

the message is deleted and the next one is printed.

no

n

the message is not deleted and the next one is printed.

reprint

print

pr

p

the message just printed is printed again, and the question is asked again.

quit

q

the user is returned to command level after the specified messages are deleted.

abort

the user is returned to command level and no messages are deleted.

---

print\_mail (prm)

---

---

print\_mail (prm)

---

### NOTES

A default mailbox is created automatically the first time a user issues print\_mail, read\_mail, accept\_messages, or print\_messages. The default mailbox is:

>user\_dir\_dir>Project\_id>Person\_id>Person\_id.mbx

To create additional mailboxes, and for more information on mailbox access, see Appendix B, "Mailbox Commands".

The user can interrupt the printing of a message by pressing the BREAK or INTERRUPT key, and then type the program\_interrupt (pi) command to proceed directly to the "Delete the message?" query. In this way, he can delete or save the message without having to print the entire message text at his terminal.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

read\_mail (rdm)

The read\_mail command provides a facility for examining and manipulating messages sent by the send\_mail and mail commands.

### SYNTAX AS A COMMAND

rdm {mbx\_specification} {-ca}

### ARGUMENTS

#### mbx\_specification

specifies the mailbox to be examined. If not specified, the user's default mailbox (>udd>Project>Person>Person.mbx) is assumed. The mailbox must be specified in one of the following forms:

-log

specifies the user's logbox and is equivalent to:

-mailbox >udd>Project\_id>Person\_id>Person\_id.sv.mbx

-mailbox PATH

-mbx PATH

specifies the pathname of a mailbox. The .mbx suffix is assumed if it is not present.

-save PATH

-sv PATH

specifies the pathname of a savebox. The .sv.mbx suffix is assumed.

-user Person\_id.Project\_id

specifies the given user's default mailbox. This control argument is equivalent to:

-mailbox >udd>Project\_id>Person\_id>Person\_id.sv.mbx

#### STR

is any non-control argument. First it is interpreted as -mailbox STR; if no mailbox is found, it is interpreted as -save STR; if no savebox is found, it is interpreted as -user STR.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

### CONTROL ARGUMENTS

-abbrev

-ab

enables abbreviation expansion of request lines.

-acknowledge

-ack

acknowledges messages that request acknowledgement. This is the default.

-brief

-bf

shortens or omits many of the informative notices printed by read\_mail.

-count

-ct

prints the total number of messages in the mailbox before entering the request loop. This is the default.

-header

-he

causes the print (pr) request to print complete message headers. This is the default.

-interactive\_messages

-im

operates on interactive messages from send\_message (when accept\_messages -hold is in effect) as well as send\_mail messages. If this control argument is not given, interactive messages are ignored.

-list

-ls

prints a summary of the messages in the mailbox before entering the request loop.

-long

-lg

prints the full text of read\_mail informative notices. This is the default.

-no\_abbrev

-na<sub>b</sub>

does not enable abbreviation expansion of request lines. This is the default.



---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

-no\_acknowledge

-nack

does not acknowledge any messages.

-no\_count

does not print the total number of messages in the mailbox before entering the request loop.

-no\_header

-nhe

causes the print (pr) request to print an abbreviated form of the message header.

-no\_interactive\_messages

-nim

operates only on send\_mail messages, not on interactive messages sent by send\_message. This is the default.

-no\_list

-nls

does not print a summary of messages before entering the request loop. This is the default.

-no\_print

-npr

does not print messages before entering the request loop. This is the default.

-no\_prompt

does not prompt for read\_mail requests when inside the request loop. This control argument is equivalent to -prompt "". The default prompt is "read\_mail(N):", where N is the recursion level if greater than one.

-no\_request\_loop

-nrql

does not enter the request loop if there are no messages in the mailbox. This is the default.

-own

operates only on the user's own messages instead of on all the messages. This control argument can be useful when examining another user's mailbox.

-print

-pr

prints the messages in the mailbox before entering the request loop.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

-profile path

-pf path

specifies the pathname of the profile to use for abbreviation expansion. The profile must already exist. The suffix "profile" is added if necessary. This control argument implies -abbrev.

-prompt STR

changes the prompt for read\_mail request lines to STR. If STR is "", the user is not prompted. STR can be an ioa\_ control string.

-quit

exits after performing any operations specified by control arguments. The default is to enter the request loop.

-request STR

-rq STR

provides an initial request line, specified by STR, to be executed by read\_mail before entering the request loop. STR must be enclosed in quotation marks if it contains blanks. Thus, the command line:

```
! read_mail -rq "print last;quit" -brief
```

prints the last message in the user's mailbox and returns to command level.

-request\_loop

-rql

enters the read\_mail request loop even if there are no messages in the mailbox.

-totals

-tt

prints the number of messages in the mailbox, and returns to command level. This control argument is incompatible with -list, -quit, -request, -request\_loop, and -print.

The -header and -no\_header control arguments can be used to set default values for the print request.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

The following control arguments can be used to set default values for the reply request:

-fill, -fi	-line_length N, -ll N
-include_authors, -iat	-no_fill, -nfi
-include_original, -io	-no_include_authors, -niat
-include_recipients, -irc	-no_include_original, -nio
-include_self, -is	-no_include_recipients, -nirc
-indent N, -in N	-no_include_self, -nis

These control arguments are described with the reply request below, with the exception of -fill, -no\_fill, and -line\_length N, which are described in the send\_mail description.

### NOTES

Many of the read\_mail requests take the same arguments and control arguments, in the form of message specifiers (spec) and message selection control arguments (-selca). These message specifiers, and selection control arguments are completely described in the next few pages; they are simply listed in the subsequent read\_mail request descriptions.

### Message Specifiers

Most read\_mail requests are capable of processing several messages in one invocation. The messages are identified by one or more message specifiers.

selection contr Message specifiers normally refer only to the messages in a mailbox that have not been marked for deletion. Most read\_mail requests accept the following control arguments, which modify the set of messages available for selection by the message specifiers:

-include_deleted	
-idl	includes all messages in the mailbox, whether or not they have been deleted, when interpreting the message specifiers to determine which messages to process.
-only_deleted	
-odl	includes only those messages that have been deleted.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

-only\_non\_deleted

-ondl

includes only those messages that have not been deleted.  
This is the default.

If a message specifier identifies a range of messages (see below), at least one message in that range must be of the appropriate type, as determined by the above control arguments.

The simplest form of a message specifier is simply a message number, such as 3. Message numbers are assigned by read\_mail when it first reads the mailbox. Even when messages are deleted, message numbers do not change during the invocation. The following keywords can be used to refer to an individual message without specifying its message number:

first

f

identifies the first message of the appropriate type in the mailbox. (The first message (#1) is identified if -idl is given; the first deleted message is identified if -odl is given; and the first non-deleted message is given if -ondl or none of these control arguments is given.)

last

l

identifies the last message of the appropriate type in the mailbox.

next

n

identifies the next message of the appropriate type in the mailbox.

previous

p

identifies the previous message of the appropriate type in the mailbox.

current

c

refers to the current message. The current message is initially the first message in the mailbox. Most requests set the current message to the last message processed by the request. For example, after executing the request:

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

print 4 12

the current message is message #12.

Ranges of messages can be identified by two message numbers or keywords separated by a colon (:). For example, the following line:

3:last

identifies all messages of the appropriate type from message #3 through the last message of the appropriate type in the mailbox. The keyword "all" is accepted as shorthand for "first:last"; it identifies all messages of the appropriate type in the mailbox.

Message numbers can be added and subtracted using "+" and "-". For example, if the current message is #20, the following line:

current-5:current+10

identifies all messages of the appropriate type from message #15 through #30. As this example demonstrates, arithmetic operations are performed after any message keywords are converted to absolute numbers.

Qedx regular expressions can be used to select all messages of the appropriate type that contain a given string. The regular expression must be enclosed in slashes (/); for an explanation of the syntax of regular expressions, see the Qedx Text Editor's User Guide, Order No. CG40. If the regular expression contains spaces, horizontal tabs, quotes ("), parentheses, or brackets, the entire expression must be enclosed in quotes to avoid misinterpretation by the request line processor; any quotes within the regular expression must be doubled. For example,

"/said, ""I think/"

---

```
read_mail (rdm)
```

---

---

```
read_mail (rdm)
```

---

matches any message that contains the string:

```
said, "I think
```

A regular expression can be preceded by one of the keywords listed above to select the first, last, etc. message containing that string. Additionally, two or more regular expressions can be combined by connectors to express logical AND (&) and logical OR (|). For example, the following line:

```
last/artificial/&/intelligence/
```

specifies the last message of the appropriate type containing both of the strings "artificial" and "intelligence".

#### Message Selection Control Arguments

The list, print, print\_header, delete, and retrieve requests accept several control arguments that supply further criteria for message selection. If no message specifiers are given, all messages of the appropriate type in the mailbox are considered for selection. For example, the request line:

```
list 23:30 -from Ellery.Proj
```

lists all non-deleted messages in the mailbox from message #23 through #30 that were sent by the user Ellery.Proj.

Selection control arguments are divided into four classes -- subject selection, time selection, author selection, and recipient selection. If several control arguments from one class are provided, a message must only satisfy one of the selections in that class to be considered by the request. If control arguments from more than one class are provided, a message must satisfy one of the selections in all of these classes provided to be considered by the request. For example, the request line:

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

```
list -from Ellery.Proj -from Green.Proj -after 1/1/82
```

lists all non-deleted messages in the mailbox that were: a) sent by either Ellery.Proj or Green.Proj, and b) sent any time from January 1982 to the present. A message sent by Ellery.Proj on 23 December 1981 would not be listed by this request.

Two control arguments allow the user to determine when to ignore the distinction between upper and lower case characters when examining header fields. All selection control arguments are affected by the following two control arguments.

-case\_sensitive

-cs

causes subject selections and qedx regular expression searches for author and recipient selections to make a distinction between upper and lower case characters. This is the default.

-non\_case\_sensitive

-ncs

causes subject selections and qedx regular expression searches for author and recipient selections to ignore the distinction between upper and lower case characters.

Thus, the following request line:

```
-sj book -non_case_sensitive
```

matches a Subject field if it contains any of the strings "book", "BOOK", "Book", etc.

Subject selection control arguments may use either qedx regular expressions or literal matches. The string value (STR) supplied to these control arguments is interpreted as a qedx regular expression if it is surrounded by slashes (/); otherwise, a literal occurrence of the string must appear in the header field. If the string contains any spaces, horizontal tabs, quotes, parentheses, or brackets, it must be enclosed in quotes to avoid misinterpretation by the request line processor, and any quotes in the string must be doubled. The following line selects messages whose Subject fields start with the string "read\_mail".

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

-sj /^read\_mail/

-in\_reply\_to STR  
-in\_reply\_to /STR/  
-irt STR  
-irt /STR/  
selects any messages whose In-Reply-To field contains STR.

-subject STR  
-subject /STR/  
-sj STR  
-sj /STR/  
selects any messages whose Subject field contains STR.

Time selection control arguments apply to the date/time that the message was created, as indicated in the message's Date header field. In the following descriptions, DT, DT1, and DT2 represent date/time strings. (For details of the acceptable date/time string formats, see the Programmer's Reference manual.) In the case of -between, -after, and -before, the date/times specified are truncated to an appropriate midnight. For example:

-between 9/1/82 9/30/82

matches all messages created during the month of September 1982.

-after DT  
-af DT  
selects any messages that were created on or after the date specified by DT.

-before DT  
-be DT  
selects any messages that were created before the date specified by DT.

-between DT1 DT2  
-bt DT1 DT2  
selects any messages that were created between the dates DT1 and DT2 inclusively.



`-date DT`

`-dt DT`

selects any messages that were created on the date specified by DT.

The following time selection control arguments do not truncate the date/times specified to an appropriate midnight. Therefore, they provide finer control on the messages selected by time:

`-after_time DT`

`-aft DT`

selects any messages that were created after the date/time specified by DT.

`-before_time DT`

`-bet DT`

selects any messages that were created before the date/time specified by DT.

`-between_time DT1 DT2`

`-btt DT1 DT2`

selects any messages that were created between the date/times specified by DT1 and DT2 inclusively.

Author and recipient selection control arguments either match the individual addresses within the appropriate header field, or match the entire content of the header field as a single string using a qedx regular expression. (See the Programmer's Reference manual for a description of appropriate address syntaxes.) If the value supplied to these control arguments is surrounded by slashes, it is interpreted as a qedx regular expression to match against the entire content of the header field. Otherwise the value, which may consist of several tokens, is interpreted as an address that must exactly match one or more of the addresses in the field.

If a qedx regular expression match is requested and the string contains any spaces, horizontal tabs, quotes, parentheses, or brackets, it must be enclosed in quotes to avoid misinterpretation by the request line processor. Further, any quotes in the string must be doubled. For example:

```
-from /Green.*Proj/
```

matches any message whose From field contains the two strings "Green" and "Proj".

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

The following line matches any message with a primary recipient named "grb" on the foreign system "System-Q".

-to grb -at System-Q
----------------------

-cc address

-cc /STR/

selects any messages whose cc field either contains the specified address or matches the given qedx regular expression.

-forwarded\_to address

-forwarded\_to /STR/

-fwdt address

-fwdt /STR/

selects any messages whose Redistributed-To field either contains the specified address or matches the given qedx regular expression.

-from address

-from /STR/

-fm address

-fm /STR/

selects any messages whose From field either contains the specified address or matches the given qedx regular expression.

-recipient address

-recipient /STR/

-rcp address

-rcp /STR/

selects any messages whose To, cc, or Redistributed-To fields either contain the specified address or match the given qedx regular expression.

-reply\_to address

-reply\_to /STR/

-rpt address

-rpt /STR/

selects any messages whose Reply-To field either contains the specified address or matches the given qedx regular expression.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

-to address

-to /STR/

selects any messages whose To field either contains the specified address or matches the given regular expression.

## REQUESTS

In the following read\_mail requests descriptions, "spec" means "message\_specifier", "-selca" means "-selection\_args", and "-ca" means "-control\_args".

?

prints a multi-columnar list of the read\_mail requests.

.

prints a line identifying the current version of read mail, the current message number, the message count, the number of deleted messages, and the pathname of the mailbox being read as in:

```
read_mail 8.3:  Message #7 of 11, 3 deleted.
                Reading your mailbox
```

If abbreviation expansion of request lines is enabled, the string "(abbrev)" is included in parentheses:

```
read_mail 8.3 (abbrev):  Message #2 of 7.
                        Reading your mailbox.
```

If the recursion level is greater than one, it is included in parentheses after the (abbrev) string, if any:

```
read_mail 8.3 (abbrev) (level 2):  Message #2 of 5.
                                   >udd>X>Y>zz.sv.mbx
```

.. STR

passes a command line, specified by STR, directly to the standard command processor, without processing by the

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

read\_mail request processor. The ".." string must be the first two characters of this request line.

abbrev {-ca}

ab {-ca}

controls abbreviation processing within read mail. If invoked with no arguments, this request enables abbrev processing within read\_mail using the profile that was last used in this read\_mail invocation. If abbrev processing was not previously enabled, the profile in use at Multics command level is used; this profile is normally [home\_dir]>Person\_id.profile. (See the Commands manual for a description of abbreviation processing.)

The read\_mail subsystem also has command line control arguments (-abbrev, -no\_abbrev, and -profile) that specify the initial state of abbreviation processing within read\_mail. For instance, a Multics abbreviation could be defined to invoke the read\_mail subsystem with a default profile as follows:

```
.ab rdm do "rdm -abbrev -profile [hd]>mail_system &rf1"
```

Control arguments may be chosen from the following:

-off

specifies that abbreviations are not to be expanded.

-on

specifies that abbreviations are expanded. This is the default.

-profile path

specifies that the segment named by path is to be used as the profile segment. The suffix ".profile" is added to path if it is not present. The segment named by path must exist prior to the use of this control argument.

[abbrev]

returns "true" if abbreviation expansion of request lines is currently enabled within read\_mail, and "false" otherwise.

all {-ca}

prints the message numbers for all messages of the specified type. Control arguments for specifying the type of message numbers may be one of the following:

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

-include\_deleted

-idl

prints the numbers of all messages in the mailbox, including deleted ones.

-no\_reverse

-nr̄v

prints the message numbers in normal order (smaller numbers first). This is the default.

-only\_deleted

-odl

prints only the numbers of deleted messages.

-only\_non\_deleted

-ondl

prints only the numbers of messages that have not been deleted. This is the default.

-reverse

rv

prints the message numbers in reverse order.

[all {-ca}]

returns the message numbers, separated by spaces, of all messages of the given type. If there are no messages of that type, it returns a null string. This active request takes the same control arguments as the all request.

answer STR {-ca} request\_line

provides preset answers to questions asked by another request. It establishes an on unit for the condition command\_question, and then executes the designated request line. If any request in the request line calls the command\_query\_subroutine (described in the Subroutines manual) to ask a question, the on unit is invoked to supply the answer. The on unit is reverted when the answer request returns to read\_mail request level. See the Reference manual for a discussion of the command\_question condition. If a question is asked that requires a yes or no answer, and the preset answer is neither "yes" nor "no", the on unit is not invoked.

The last answer specified is issued as many times as necessary, unless followed by the -times N control argument.

The -match and -exclude control arguments are applied in the order specified. Each -match causes a given question to be answered if it matches STR; each -exclude causes it to be passed on if it matches STR. A question that has been

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

excluded by the `-exclude` control argument is reconsidered if it matches a `-match` later in the request line.

The arguments are:

**STR**

is the desired answer to any question. If the answer is more than one word, it must be enclosed in quotes. If STR is `-query`, the question is passed on to the user. The `-query` control argument is the only one that can be used in place of STR.

**request\_line**

is any `read_mail` request line. It can contain any number of separate arguments (i.e., have spaces within it) and need not be enclosed in quotes.

Control arguments may be chosen from the following:

**-brief**

**-bf**

suppresses printing (on the user's terminal) of both the question and the answer.

**-call STR**

evaluates the active string STR to obtain the next answer in a sequence. The active string is constructed from `read_mail` active requests and Multics active strings (using `read_mail`'s "execute" active request). The outermost level of brackets must be omitted and the entire string must be enclosed in quotes if it contains request processor special characters. The return value "true" is translated to "yes", and "false" to "no". All other return values are passed as is.

**-exclude STR**

**-ex STR**

passes on, to the user or other handler, questions whose text matches STR. If STR is surrounded by slashes (/), it is interpreted as a `qedx` regular expression. Otherwise, answer tests whether STR is literally contained in the text of the question. Multiple occurrences of `-exclude` are allowed; they apply to the entire request line.

**-match STR**

answers only questions whose text matches STR. If STR is surrounded by slashes (/), it is interpreted as a `qedx` regular expression. Otherwise, answer tests whether STR is literally contained in the text of the question.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

Multiple occurrences of -match are allowed; they apply to the entire request line.

-query

skips the next answer in a sequence, passing the question on to the user. The answer is read from the user\_i/o I/O switch.

-then STR

supplies the next answer in a sequence.

-times N

gives the previous answer (STR, -then STR, or -query) N times only (where N is an integer).

append {spec} path {-ca}

app {spec} path {-ca}

appends the specified messages (with headers) to the ASCII segment specified by path. The suffix .mail is added to path if it is not present. If the specified segment does not already exist, the user is asked whether to create it. This request causes the specified messages to be acknowledged, if requested by the senders (see send\_mail -acknowledge). If required, it adds Date and From fields to the ASCII representations of the messages it places into the segment. Control arguments are:

-delete

-dl

deletes the messages after appending them, if all the append operations were successful.

-include\_deleted

-idl

writes all specified messages, including deleted ones.

-no\_delete

-ndl

does not delete the messages after appending them. This is the default.

-no\_reverse

-nr̄v

writes the messages in ascending numeric order. This is the default.

-only\_deleted

-odl

writes only deleted messages.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

-only\_non\_deleted

-ondl

writes only those messages that have not been deleted.  
This is the default.

-reverse

-rev

appends the messages in reverse order.

apply {spec} {-ca} STR

ap {spec} {-ca} STR

places the text of the selected message(s) into a temporary segment in the process directory, then concatenates the command line specified by STR with intervening spaces and appends the pathname of the temporary segment. This command line is passed to the Multics command processor. The command line may not modify the contents of the temporary segment. Each message is processed individually. For example, the following read\_mail request line:

```
apply /Gomez/ "do ""copy &1 &!; eor &! -dl"""
```

issues a separate output request for each message containing the string "Gomez".

The supplied command line need not be enclosed in quotes. However, if (), [], or " are in the command line to be processed by the Multics command processor, they should be enclosed in quotes to prevent processing by read\_mail's request processor. Control arguments are:

-delete

-dl

deletes the messages after processing them, if all messages are successfully processed.

-header

-he

specifies that the header of each message is to be included in the temporary segment. This is the default.

-include\_deleted

-idl

processes all specified messages, including deleted ones.



---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

**-no\_delete**

**-ndl**

does not delete the messages after processing them. This is the default.

**-no\_header**

**-nhe**

specifies that the header of each message is not to be included in the temporary segment.

**-no\_reverse**

**-nrv**

processes the messages in ascending numeric order. This is the default.

**-no\_text**

specifies that the text of each message is not to be included in the temporary segment.

**-only\_deleted**

**-odl**

processes only deleted messages.

**-only\_non\_deleted**

**-ondl**

processes only those messages that have not been deleted. This is the default.

**-reverse**

**-rev**

processes the messages in reverse order.

**-text**

specifies that the text of each message is included in the temporary segment.

**copy {spec} path {-ca}**

**cp {spec} path {-ca}**

copies the specified messages into the mailbox designated by path. The mailbox must already exist. The .mbx suffix is added to path if it is not present. The messages are copied exactly as they appear in the original mailbox; no header fields are added, interactive messages are not converted to normal messages, etc. This request does not send acknowledgements for any of the messages that it processes. If the original message requests an acknowledgement, the copied message also requests an acknowledgement to the sender of the original message. Control arguments are the same as for the append request.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

current

c

prints the number of the current message.

[current]

returns the number of the current message, or 0 if there is no current message.

delete {spec} {-selca} {-ca}

dl {spec} {-selca} {-ca}

d {spec} {-selca} {-ca}

deletes the specified messages. If no messages are specified, the current one is deleted. Deleted messages can be retrieved before exiting read\_mail by using the retrieve (rt) request. The user is queried for permission if he attempts to delete a message that has not been the subject of one of the following requests: apply, copy, forward, list, log, preface, print, print\_header, reply, save, write. Thus the user is protected from accidentally deleting newly-arrived messages without having first examined them.

Control arguments for the delete request may be one of the following:

-force

-fc

deletes unprocessed messages without querying, and ignores messages that can not be deleted due to insufficient access.

-no\_force

-nfc

queries the user for permission to delete any unprocessed messages. No message is deleted if either the user answers "no" to a query, or the user lacks sufficient access to delete one or more of the specified messages.

do STR {args}

or

do {-ca}

expands a request line specified by STR by substituting the supplied arguments into the line before execution. Arguments are character string arguments that replace parameters in the request line.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

The following control arguments set the mode of operation of the do request:

**-absentee**

an any\_other handler is established that catches all conditions and aborts execution of the request line without aborting the process.

**-brief**

**-bf**

the expanded request line is not printed before execution. This is the default.

**-go**

the expanded request line is passed on for execution. This is the default.

**-interactive**

the any\_other handler is not established. This is the default.

**-long**

**-lg**

the expanded request line is printed before execution.

**-nogo**

the expanded request line is not passed on for execution.

Any sequence beginning with & in the request line is expanded by the do request using the arguments given on the request line. Following is the list of parameters:

**&I**

is replaced by argI. I must be a digit from 1 to 9.

**&(I)**

is also replaced by argI. I can be any value, however.

**&qI**

is replaced by argI with any quotes in argI doubled. I must be a digit from 1 to 9.

**&q(I)**

is also replaced by argI with any quotes doubled. I can be any value.

**&rI**

is replaced by all the arguments starting with argI. Each argument is placed in quotes with contained quotes doubled. I must be a digit from 1 to 9.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

&r(I)  
is also replaced by a requoted argI. I can be any value.

&fI  
is replaced by all the arguments starting with argI. I must be a digit from 1 to 9.

&f(I)  
is also replaced by all the arguments starting with argI. I can be any value.

&qfI  
is replaced by all the arguments starting with argI with any quotes doubled. I must be a digit from 1 to 9.

&qf(I)  
is also replaced by all the arguments starting with argI with quotes doubled. I can be any value.

&rf(I)  
is also replaced by all the arguments starting with argI, requoted. I can be any value.

&&  
is replaced by an ampersand.

&!  
is replaced by a 15 character unique string. The string used is the same everywhere &! appears in the request line.

&n  
is replaced by the actual number of arguments supplied.

&f&n  
is replaced by the last argument supplied.

[do STR {args}]  
returns a request line specified by STR with argument substitution.

exec\_com path {args}  
ec path {args}

executes a program written in the exec\_com language, where path is the pathname of an exec\_com program. The suffix "rdmec" is added to the pathname if necessary. This program is used to pass request lines to read\_mail and to pass input lines to requests that read input. Currently, any errors detected during an ec execution within read\_mail will abort the request line in which the ec request was invoked. The

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

arguments are optional arguments to the exec\_com program and are substituted for parameter references in the program such as &1.

If the pathname does not contain a "<" or ">" character, read\_mail searches for the exec\_com program using the mail\_system search list. The default content of this search list is:

<pre>-working_dir &gt;udd&gt;[user project]&gt;[user name]&gt;[user name].mlsys</pre>
---

When evaluating a read\_mail exec\_com program, subsystem active requests are used rather than Multics active functions when evaluating the &[...] construct and the active string in an &if statement. The read\_mail execute active request may be used to evaluate Multics active strings within the exec\_com.

[exec\_com path {args}]

[ec path {args}]

executes a program written in the exec\_com language that specifies a return value of the exec\_com request by use of the &return statement. The arguments are the same as for the exec\_com request.

execute STR

e STR

executes the supplied line as a Multics command line, where STR is the Multics command line to be executed or the Multics active string to be evaluated. It need not be enclosed in quotes.

The recommended method to execute a Multics command line from within read\_mail is the ".." escape sequence. The execute request is intended as a means of passing information from read\_mail to the Multics command processor.

All (), [], and ''s in the given line are processed by the read\_mail request processor, not the Multics command processor. Thus, the values of subsystem active requests may be passed to Multics commands when using the execute request. For example, the following request line lists the ACL of the mailbox being read by the current invocation of read\_mail.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

```
e mbla [mailbox]
```

[execute STR]

[e STR]

evaluates a Multics active string from within read\_mail. For example, the following read\_mail request line:

```
write all [e strip_entry [mailbox]]
```

writes the ASCII representation of all messages in the mailbox into a segment in the working directory whose entry name is the same as that of the mailbox, with the "mbx" suffix changed to "mail".

first {-ca}

f {-ca}

prints the number of the first message of the specified type. The control argument may be one of the following:

-include\_deleted

-idl

prints the number "1" (i.e., the number of the first message, whether or not it has been deleted.)

-only\_deleted

-odl

prints the message number of the first deleted message.

-only\_non\_deleted

-ondl

prints the message number of the first non-deleted message. This is the default.

[first {-ca}]

[f {-ca}]

returns the number of the first message of the specified type. If there are no messages of the specified type, it returns the value zero. This active request takes the same control arguments as the first request.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

forward {spec} addresses {-ca}

fwd {spec} addresses {-ca}

for {spec} addresses {-ca}

forwards the specified message to the addresses given. Control arguments to the forward request are the same as those for the append request.

Forwarding addresses may be given in any of the forms described under "Addresses" in the send\_mail command description (later in this appendix).

This request adds three fields to the message header: Redistributed-Date, Redistributed-By, and Redistributed-To. It adds the Date and From fields to the original message if necessary before forwarding. The request also causes the message to be acknowledged, if requested by the original sender (see send\_mail -acknowledge).

To forward a set of messages that can not be identified by a single message specifier, request line iteration and the list active request may be used to avoid retyping the recipients. For example:

```
forward ([list 1 3 9 last-4:last]) Fry.ABC Lee.Proj -dl
```

help {STR} {-ca}

prints information about various read\_mail topics, including detailed descriptions of read\_mail requests. If specified, STR is the name of a read\_mail request or one of the other available topics. If STR is not specified, the help request lists the requests that provide information about read\_mail.

The help request accepts most of the control arguments accepted by the Multics help command. Type ".. help help" for a complete description of the help request. Following is a description of some of the more useful control arguments for the help request:

-brief

-bf

prints only a summary of a request or active request, including the Syntax section, list of arguments, control arguments, etc.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

-search STRs

-srh STRs

begins printing with the paragraph containing all the strings STRs. By default, printing starts at the beginning of the information.

-section STRs

-scn STRs

begins printing at the section whose title contains all the strings STRs. By default, printing starts at the beginning of the information.

-title

prints section titles and section line counts; then asks if the user wants to see the first paragraph of information.

The most useful responses to questions asked by the help request are:

?

prints the list of responses allowed to help queries.

.

prints "help" to identify the current interactive environment.

.. command\_line

treats the remainder of the response as a Multics command line.

no

n

stops printing information for this topic and proceeds to the next topic, if any.

quit

q

stops printing information for this topic and returns to the subsystem's request level.

rest {-scn}

r {-scn}

prints remaining information for this topic without intervening questions. If -section or -scn is given, help prints only the rest of the current section without questions and then asks if the user wants to see the next section.



search {STRs} {-top}  
srh {STRs} {-top}  
skips to the next paragraph containing all the strings STRs. If -top or -t is given, searching starts at the top of the information. If STRs are omitted, help uses the STRs from the previous search response or the -search control argument.

section {STRs} {-top}  
scn {STRs} {-top}  
skips to the next section whose title contains all the strings STRs. If -top or -t is given, title searching starts at the top of the information. If STRs are omitted, help uses the STRs from the previous section response or the -section control argument.

skip {-scn} {-seen}  
s {-scn} {-seen}  
skips to the next paragraph. If -section or -scn is given, help skips all paragraphs of the current section. If -seen is given, help skips to the next paragraph that the user has not seen. Only one control argument is allowed in each skip response.

title {-top}  
lists titles and line counts of the sections that follow; if -top or -t is given, help lists all section titles. The previous question is repeated after titles are printed.

yes  
Y  
prints the next paragraph of information on this topic.

if [EXPR] -then LINE1 {-else LINE2}  
conditionally executes one of two request lines depending on the value of an active string. The arguments are:

EXPR  
is the active string that must evaluate to either "true" or "false". The active string is constructed from read\_mail active requests and Multics active strings (using read\_mail's execute active request).

LINE1  
is the read\_mail request line to execute if EXPR evaluates to "true". If the request line contains any request processor characters, it must be enclosed in quotes.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

LINE2

is the read\_mail request line to execute if EXPR evaluates to "false". If omitted and EXPR is "false", no additional request line is executed. If the request line contains any request processor characters, it must be enclosed in quotes.

[if [EXPR] -then STR1 {-else STR2}]

returns one of two character strings to the read\_mail request processor, depending on the value of an active string. The arguments are:

EXPR

is the active string that must evaluate to either "true" or "false". The active string is constructed from read\_mail active requests and Multics active strings (using read\_mail's execute active request).

STR1

is returned as the value of the if active request if the EXPR evaluates to "true".

STR2

is returned as the value of the if active request if the EXPR evaluates to "false". If omitted and the EXPR is "false", a null string is returned.

last {-ca}

l {-ca}

prints the number of the last message of the specified type. The control argument may be one of the following:

-include\_deleted

-idl

prints the number of the last message, whether or not it has been deleted.

-only\_deleted

-odl

prints the number of the last deleted message.

-only\_non\_deleted

ondl

prints the message number of the last non-deleted message. This is the default.

[last {-ca}]

[l {-ca}]

returns the number of the last message of the specified type. If there is no message of the specified type, it returns the

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

value zero. This active request takes the same control arguments as the last request.

list {spec} {-selca} {-ca}

ls {spec} {-selca} {-ca}

prints a summary line for each of the specified messages, or for all undeleted messages if no specifiers are given. Control arguments may be chosen from the following:

-delete

-dl

deletes the messages after listing them.

-header

-he

prints a header line before the list of messages. This is the default.

-include\_deleted

-idl

prints the list of messages, including deleted ones.

-line\_length N

-ll N

prints the list of messages, using the supplied line length N to determine where and if to truncate the message subject. (The default length is the terminal's line length.)

-no\_delete

-ndl

does not delete the messages after listing them. This is the default.

-no\_header

-nhe

omits the header line preceding the list of messages.

-no\_line\_length

-nll

does not truncate the message subject unless the subject is more than one line long.

-no\_reverse

-nrv

lists the messages in ascending numeric order. This is the default.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

-only\_deleted

-odl

lists only deleted messages.

-only\_non\_deleted

-ondl

lists only non-deleted messages. This is the default.

-reverse

-rev

prints the list of messages in reverse order.

The current message is marked (in the listing) by a "\*" to the right of the message number. If -idl or -odl is specified, deleted messages are marked by an "!" to the right of the message number.

One or two lines are printed for each message. The format of the first line is:

N	(L)	MM/DD/YY	HH:MM	AUTHOR	SUBJECT
---	-----	----------	-------	--------	---------

where N is the message number and L is the number of lines in the body of the message (excluding the header). MM/DD/YY HH:MM specifies the date/time when the message was originally transmitted. AUTHOR specifies the original author(s) of the message, and is normally as much of the From field of the message as will fit in the provided space. SUBJECT is as much of the Subject field, if present, as will fit on the line. If the message is an interactive message, SUBJECT is as much of the actual text of the message as will fit on the line.

If the message has been forwarded, a second line is included in the listing. This line has the format:

(*) Forwarded (Nth time) at MM/DD/YY HH:MM by STR
---

where N indicates the number of times that this message has been forwarded. (N is omitted if the message has only been forwarded once.) MM/DD/YY HH:MM specifies the date/time that the message was last forwarded, and is derived from the most recent Redistributed-Date field. STR specifies the person

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

who last forwarded the message, and is the contents of the most recent Redistributed-By field in the message.

[list {spec} {-selca} {-ca}]

[ls {spec} {-selca} {-ca}]

returns a list of the numbers of the specified messages separated by spaces. This active request takes the same selection arguments and control arguments as the list request.

list\_help {topics}

lh {topics}

displays the name of all read\_mail information segments on given topics. If no topics are given, all read\_mail information segments are listed.

When matching topics with info segment names, an info segment name is considered to match a topic only if that topic is at the beginning or end of a word within the segment name. Words in info segment names are bounded by the beginning and end of the segment name and by the characters period (.), hyphen (-), underscore (\_), and dollar sign (\$). The ".info" suffix is not considered when matching topics.

list\_requests {STR} {-ca}

lr {STR} {-ca}

prints a brief description of selected read\_mail requests, where STR specifies the request(s) to be described. Any request with a name containing one of these strings is listed unless -exact is used, in which case the request name must exactly match one of these strings. When matching STRs with request names, a request name is considered to match a STR only if that STR is at the beginning or end of a word within the request name. Words in request names are bounded by the beginning and end of the request name and by the characters period (.), hyphen (-), underscore (\_), and dollar sign (\$).

Control arguments are:

-all

-a

includes undocumented and unimplemented requests in the list of requests eligible for matching the STR arguments.

-exact

lists only those requests one of whose names exactly match one of the STR arguments.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

log {spec} {-ca}  
saves the specified messages in the user's logbox. The user's logbox has the pathname >udd>Project\_id>Person\_id>Person\_id.sv.mbx. It is created automatically if it does not already exist, and the user is informed of its creation. Date and From header fields are added as required to logged messages. Any messages requiring acknowledgement are acknowledged unless -no\_acknowledge is specified on the read\_mail command line. Control arguments for this request are the same as for the append request.

mailbox

mbx

prints the absolute pathname of the mailbox currently being read.

[mailbox]

[mbx]

returns the absolute pathname of the mailbox currently being read.

next {-ca}

prints the number of the next message of the specified type. The control argument may be one of the following:

-include\_deleted

-idl

prints the number of the next message in the mailbox, whether or not it has been deleted.

-only\_deleted

-odl

prints the number of the next deleted message.

-only\_non\_deleted

-ondl

prints the number of the next non-deleted message. This is the default.

[next {-ca}]

returns the number of the next message number of the specified type. If there are no messages of the specified type, the value zero is returned. This active request takes the same control arguments as the next request.

preface {spec} path {-ca}

prf {spec} path {-ca}

same as the append request, but inserts messages at the beginning of the ASCII segment specified by path, rather than at the end.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

previous {-ca}

prints the number of the previous message of the specified type. The control argument may be one of the following:

-include\_deleted

-idl

prints the number of the previous message, whether or not it has been deleted.

-only\_deleted

-odl

prints the number of the previous deleted message.

-only\_non\_deleted

-ondl

prints the number of the previous non-deleted message. This is the default.

[previous {-ca}]

returns the number of the previous message of the specified type. If there is no message of the specified type, the value zero is returned. This active request takes the same control arguments as the previous request.

print {spec} {-selca} {-ca}

pr {spec} {-selca} {-ca}

p {spec} {-selca} {-ca}

prints the specified messages. This request causes the specified messages to be acknowledged, if requested by the sender, unless -no\_acknowledge is specified on the read\_mail command line.

If you use this request while in the video system (documented in The Multics Menu System, Order No. CP51), the reset\_more control order is issued after each message is printed. This allows users of the video system to easily abort the printing of a single message, when printing several messages.

Control arguments may be chosen from the following:

-delete

-dl

deletes the specified messages upon exiting read\_mail, if all the specified messages are successfully printed.

-header

-he

prints complete message headers with the message text. This is the default.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

-include\_deleted

-idl

prints the messages, whether or not they have been deleted.

no\_delete

-ndl

does not delete the specified messages upon exiting read\_mail. This is the default.

-no\_header

-nhe

prints an abbreviated form of the message header.

-no\_reverse

-nrv

prints the messages in ascending numeric order. This is the default.

-only\_deleted

-odl

prints only the deleted messages.

-only\_non\_deleted

-ondl

prints the non-deleted messages. This is the default.

-reverse

-rev

prints messages in reverse order.

print\_header {spec} {-selca} {-ca}

prhe {spec} {-selca} {-ca}

prints only the header of the specified message. This request causes the specified messages to be acknowledged if requested by the sender, unless -no\_acknowledge is specified on the read\_mail command line. Control arguments are the same as for the print request, except that the print\_header request does not take the -header and -no\_header control arguments.

quit {-ca}

q {-ca}

exits the read\_mail command; any requested deletions are actually performed at this point. Control arguments may be chosen from the following:



---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

-delete  
-dl  
deletes the specified messages upon exiting read\_mail.  
This is the default.

-force  
-fc  
does not check for newly arrived messages before  
returning to command level.

no\_delete  
-ndl  
does not delete the specified messages upon exiting  
read\_mail.

-no\_force  
-nfc  
queries the user for permission to exit read\_mail if  
there are newly arrived messages. This is the default.

ready  
rdy  
prints a Multics ready message. The Multics general\_ready  
command may be used to change the format of the ready message  
printed by this request, and also after execution of request  
lines if the ready\_on request is used. The default ready  
message gives the time of day, the amount of CPU time, and  
page faults used since the last ready message was typed.

ready\_off  
rdf  
does not generate a ready message after the execution of each  
request line. This is the default.

ready\_on  
rdn  
causes a ready message to be printed after the execution of  
each request line.

reply {spec} {-ca} {-to addresses} {-ca more\_addresses}  
rp {spec} {-ca} {-to addresses} {-ca more\_addresses}  
allows the user to reply to the specified messages. By  
default, the reply is sent only to the authors of the  
original messages. The reply is created in send\_mail; the  
user is returned to read\_mail after the message is sent.  
This request acknowledges any messages requiring  
acknowledgement unless -no\_acknowledge is specified on the  
read\_mail command line.

Control arguments for the reply request are:

- cc addresses**  
sends a copy of the reply to the specified addresses. The given addresses become the only secondary recipients of the reply unless the `-include_recipients` control argument is also included.
- delete**  
**-dl**  
deletes the messages after replying to them. However, if you exit `send_mail` without sending the reply, this control argument is ignored.
- include\_authors**  
**-iat**  
includes the authors of the original messages as primary recipients of the new message. This is the default, unless `-to` is also specified, in which case this argument must be explicitly specified if the authors are to receive the reply.
- include\_deleted**  
**-idl**  
includes all messages in the mailbox, whether or not they have been deleted, when processing the `message_specifiers` to determine which messages will be answered.
- include\_original**  
**-io**  
includes the text and the Date, From, and Subject fields of the messages being replied to in the reply. This text is indented four spaces if no indentation is explicitly specified.
- include\_recipients**  
**-irc**  
includes all recipients of the original message as secondary recipients of the reply.
- include\_self**  
**-is**  
allows a copy of the reply to be sent to the writer of the reply if it is determined that such a copy should be sent from the use of the `-include_authors` or `-include_recipients` control arguments.
- indent N**  
**-ind N**  
indents the text of the original message by N spaces in the reply. The default is 4 spaces.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

-no\_delete

-ndl

does not delete the messages. This is the default.

-no\_include\_authors

-niat

does not include the authors of the message as primary recipients.

-no\_include\_original

-nio

does not include the original messages. This is the default.

-no\_include\_recipients

-nirc

does not include the recipients of the message as secondary recipients. This is the default.

-no\_include\_self

-nis

specifies that a copy of the reply is sent to the writer of the reply only if this is explicitly requested by use of the -to or -cc control arguments. This is the default. This default allows the user to create a reply abbreviation that automatically logs the reply without receiving an extra copy whenever -include\_recipients is specified.

-no\_refill

-nrfi

does not reformat the original text. This is the default.

-only\_deleted

-odl

includes only deleted messages when processing the message\_specifiers to determine which messages will be answered.

-only\_non\_deleted

-ondl

includes only non-deleted messages when processing the message\_specifiers to determine which messages will be answered. This is the default.

-refill

-rfi

reformats the original text to fit within the line length of the reply.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

**-to addresses**

sends a copy of the reply to the specified addresses. The -to control argument overrides the -include\_authors default, so the given addresses become the only primary recipients of the reply unless the -include\_authors control argument is also included.

The following send\_mail control arguments can also be used on the reply request line:

-abbrev, -ab	-no_fill, -nfi
-abort	-no_log
-acknowledge, -ack	-no_message_id, -nmid
-brief, -bf	-no_prompt
-fill, -fi	-no_request_loop, -nrql
-from addresses	-no_subject, -nsj
-input_file path, -if path	-profile_path, -pf path
-line_length N, -ll N	-prompt STR
-log	-reply_to addr, -rpt addr
-long, -lg	-request STR, -rq STR
-message_id, -mid	-request_loop, -rql
-no_abbrev, -nab	-save_path, -sv path
-no_abort	-subject STR, -sj STR
-no_acknowledge, -nack	-terminal_input, -ti

(For the -reply\_to control argument in the above list, "addr" means "addresses".)

**Notes on recipients:**

By default, the reply is sent only to the authors of the original messages or to those recipients specified by the authors to receive replies in place of the authors. In the following text, the term "authors of the original messages" means either the authors or their designated agents.

The -to and -include\_authors control arguments specify the primary recipients for the reply. If the -to control argument is used and -include\_authors does not appear on the request line, only those addresses specified after -to are used as the primary recipients of the reply. If both -to and -include\_authors are used on the request line, the primary recipients of the message are the authors of the original messages and the addresses specified after the -to control argument. Use of -include\_authors on the read\_mail command line does not affect this interaction of -to and -include\_authors on the reply request line.

The -cc and -include\_recipients control arguments specify the secondary recipients for the reply. If -include\_recipients is specified either on the reply request line or the

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

read\_mail command line, all recipients of the original messages are included as secondary recipients of the reply. If -cc is used on the request line, the addresses following the -cc control argument are added to the list of secondary recipients of the reply. For example, the command line:

```
read_mail -include_recipients
```

in conjunction with the request line

```
reply -to Smith.Proj -cc Riley.Proj
```

composes a reply for the current message that is sent to Smith.Proj as the sole primary recipient and to all the recipients of the current message plus Riley.Proj as the secondary recipients.

Notes:

Unless overridden by use of the -abbrev, -no\_abbrev, or -profile control arguments, the send\_mail invocation created by this request has the same state of request line abbreviation expansion and uses the same profile as the current read\_mail invocation.

Unless overridden by use of the -subject or -no\_subject control arguments, this request constructs a subject for the reply message by combining the subjects of all the original messages. Additionally, the subject is prefixed by the string "Re: ".

This request constructs an In-Reply-To field for the reply message identifying the original messages being answered by this reply.

retrieve {spec} {-selca}

rt {spec} {-selca}

causes the specified messages, if deleted, to be undeleted. This action is allowed until the user quits and returns to command level. When the user exits read\_mail, all messages deleted by the delete (dl) request are actually deleted from the mailbox and can no longer be retrieved.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

save {spec} path {-ca}

sv {spec} path {-ca}

saves the specified messages in the mailbox designated by path. The .sv.mbx suffix is added to path if it is not present. If the savebox does not exist, the user is asked whether to create it. Date and From fields are automatically added to any messages that do not have them. If no messages are specified, the current one is saved. This request causes the specified messages to be acknowledged if requested by the senders, unless -no\_acknowledge is specified on the read\_mail command line. Control arguments are the same as for the append request.

subsystem\_name

prints the name of the current subsystem.

[subsystem\_name]

returns the name of the current subsystem. This active request is useful as part of an abbrev that is shared by multiple subsystems.

subsystem\_version

prints the version of the current subsystem.

[subsystem\_version]

returns the version of the current subsystem. This active request may be used in an abbrev that is shared by multiple subsystems.

write {spec} path {-ca}

w {spec} path {-ca}

appends the specified messages to the ASCII segment designated by path. The .mail suffix is added to path if it is not present. If no messages are specified, the current one is written. Date and From fields are added to any messages that do not have them. This request causes the specified messages to be acknowledged if requested by the senders unless -no\_acknowledge is specified on the read\_mail command line. Control arguments may be chosen from the following:

-delete

-dl

deletes the messages after writing them, if all the write operations are successful.

-extend

writes the messages at the end of the segment. This is the default.

---

read\_mail (rdm)

---

---

read\_mail (rdm)

---

-include\_deleted  
-idl  
writes the messages, whether or not they have been deleted.

-no\_delete  
-ndl  
does not delete the messages after writing them. This is the default.

-no\_reverse  
-nrv  
writes the messages in ascending numeric order. This is the default.

-only\_deleted  
-odl  
writes only the deleted messages.

-only\_non\_deleted  
-ondl  
writes the non-deleted messages. This is the default.

-reverse  
-rev  
writes the messages in reverse order.

-truncate  
-tc  
truncates the segment before writing the messages to it.

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

send\_mail (sdm)

The send\_mail command transmits a message to one or more recipients. The message is automatically prefixed by a header whose standard fields give the author(s), the intended recipients, and a brief summary of the contents.

#### SYNTAX AS A COMMAND

sdm {addresses} {-ca}

#### ARGUMENTS

##### addresses

specifies the primary recipients of the message. By default, the message has no primary recipients. Addresses may be specified in one or more of the following forms:

-log

specifies the user's logbox and is equivalent to:

-mailbox >udd>Project\_id>Person\_id>Person\_id.sv.mbx

-mailbox PATH

-mbx PATH

specifies the pathname of a mailbox. The .mbx suffix is assumed if it is not present.

-save PATH

-sv PATH

specifies the pathname of a savebox. The .sv.mbx suffix is assumed.

-user Person\_id.Project\_id

specifies the given user's default mailbox. This control argument is equivalent to:

-mailbox >udd>Project\_id>Person\_id>Person\_id.sv.mbx

STR

is any non-control argument. If STR contains either "<" or ">", it is interpreted as -mailbox STR. Otherwise, it is interpreted as -user STR.

STR -at System

's valid only on systems connected to the ARPA network and specifies an address on another computer system. STR



---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

identifies the user (or group of users) to receive the message; it is not interpreted in any way by the local system. System identifies a remote system defined in the local system's host table; no distinction is made between upper and lower case characters in the host name.

-comment STR

-com STR

must appear immediately following one of the above forms of an address; it supplies additional descriptive information about the address such as a user's full name.

### CONTROL ARGUMENTS

Control arguments can be interspersed with the addresses and can be chosen from the following:

-abbrev

-ab

enables abbreviation expansion of request lines.

-abort

does not send the message unless it can be successfully delivered to all specified recipients. This is the default.

-acknowledge

-ack

requests that a message be sent to the user of send\_mail by each recipient of the message after they have read the message via read\_mail or print\_mail. The sender's name is placed in the Acknowledge-To header field.

-brief

-bf

suppresses printing of the message "Mail delivered to <address>" when mail is sent.

-cc addresses

adds subsequent addresses as secondary recipients of the message. Mail is sent to these addresses when the send request is issued with no arguments. These addresses are placed in the cc header field. In the default case there are no secondary recipients.

-fill

-fi

reformats the text of the message according to "fill-on" and "align-left" modes of the compose command. The message is

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

reformatted after initial input is completed, and after each execution of the qedx and apply requests. This is the default for terminal input.

-from addresses

adds subsequent addresses as authors of the message. These addresses are placed in the From header field, overriding the sender's name (placed there by default), and are used as recipients of a reply request.

-header

-he

generates a message header. This is the default.

-in\_reply\_to STR

-irt STR

places STR in the In-Reply-To field of the header. In the default case this field is not present.

-input\_file path

-if path

sends the text of a segment. Use of this control arguments implies -rql. If -input\_file is not specified, the user is prompted for the message text ("Message:").

-line\_length N

-ll N

specifies a line length to be used when adjusting text. The default line length is 72 characters.

-long

-lg

prints the "Mail delivered to <address>" message when mail is sent. This is the default.

-message\_id

-mid

adds a Message-ID field to the header, containing a unique identifier for the message.

-no\_abbrev

-nab

does not enable abbreviation expansion of request lines. This is the default.

-no\_abort

sends the message to as many recipients as possible, even if it cannot be sent to all specified recipients.

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

-no\_underscore\_underscore

-nack

does not request that recipients of the message acknowledge the message. This is the default.

-no\_fill

-nfi

sends the message as typed with no formatting adjustments, unless the fill request or the -fill control argument of the qedx and appy requests is used. This is the default for file input.

-no\_header

-nhe

does not add the normal message header to the message. Only the Subject and To header fields are added unless explicitly requested by control arguments or requests.

-no\_log

does not send a copy of the message to the user's logbox. This is the default.

-no\_message\_id

-nmid

does not add a Message-ID field to the header. This is the default.

-no\_prompt

does not prompt for request lines when inside the request loop. The default prompt is "send\_mail(N):", where N is the recursion level if greater than one.

-no\_request\_loop

-nrql

sends the message upon completion of input without entering the request loop, unless input is from a terminal and is terminated by "\f" or "\q". This is the default for terminal input.

-no\_subject

-nsj

does not add a Subject field to the header.

-profile path

-pf path

specifies the pathname of the profile to use for abbreviation expansion. The suffix "profile" is added if necessary. This control argument implies -abbrev.

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

- prompt STR**  
sets the prompt for the request loop to the ioa\_control string STR. If STR is "", the user is not prompted.
- reply\_to addresses**  
**-rpt addresses**  
adds subsequent addresses to the Reply-To header field. In the default case this field is not present. When present, these addresses are used as recipients of a reply request, rather than the addresses of the From field.
- request STR**  
**-rq STR**  
executes a line of requests specified by STR, after reading the message text from the appropriate source. If the quit (q) request is not included in STR, the request loop is entered after STR is executed.
- request\_loop**  
**-rql**  
enters send\_mail's request loop after reading the message text. This is the default for file input.
- subject STR**  
**-sj STR**  
places STR in the Subject field of the header. If STR is "", no Subject field is created. If this control argument is not specified, the user is asked for a subject with the prompt "Subject:". A blank response causes the Subject field to be omitted.
- terminal\_input**  
**-ti**  
prompts the user for the message text ("Message:"). The user then types the message text terminated by a line consisting of a period ("."). This is the default.
- to addresses**  
adds subsequent addresses as primary recipients of the message. These addresses, along with the addresses at the beginning of the command line (preceding any control arguments), are placed in the To header field. Mail is sent to these recipients when the send request is issued with no arguments. There are no primary recipients by default.

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

## NOTES

If conflicting control arguments (for instance, `-header` and `-no_header`) are specified, the last one takes effect.

## Terminal Input

By default or if `-terminal_input` is specified, `send_mail` issues the prompt "Message:" and reads the message text from the terminal.

If the user terminates the text with a line containing just a period (`.`), `send_mail` reformats the message unless `-no_fill` was given on the command line. It then sends the message to the specified recipients, unless `-request` or `-request_loop` was also given on the command line. If any errors occur while sending the message, `send_mail` enters its request loop to allow the user to correct the problem.

If the user terminates the text with a line containing `"\f"` anywhere on the line, `send_mail` enters the `gedx` editor. Any characters on the line after the `"\f"` are treated as `gedx` requests.

If the user terminates the text with a line containing `"\q"` anywhere on the line, `send_mail` reformats the message (unless `-no_fill` is given on the command line), and enters the request loop. Any characters on the line after the `"\q"` are ignored with a warning message. Type "help `gedx`" within `send_mail` for more information on the `gedx` request.

## Addresses

Any addresses appearing on the command line before the first `-cc`, `-from`, `-reply_to`, or `-to` control argument are considered primary recipients of the message. (See the description of the `-to` control argument.)

The `-cc`, `-from`, `-reply_to`, and `-to` control arguments apply to all subsequent addresses until the next of these control arguments is given. Any other intervening control arguments do not affect this interpretation.

For example, the sequence:

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

```
addr1 -from addr2 addr3 -cc addr4 -to addr5
```

causes addr1 and addr5 to be processed by -to, addr2 and addr3 to be processed by -from, and addr4 to be processed by -cc.

### Headers

Each message in a mailbox includes a header containing information about who sent the message, when the message was sent, etc. The message header is composed of header fields. Each field contains its name, a colon, and the contents of the field. The header is separated from the actual text of the message by one or more blank lines.

The following group of fields are used by the Multics mail system. Additional fields may be present in a message's header for use by subsystems that use the mail system to store and transfer information. Among the standard fields, only the Date and From fields are always present in a message; all other fields are optional. The fields are presented in the order that they actually appear in a header.

#### Date:

specifies the date and time that the message was created. Its format is:

```
Date: DOW, MM Month YYYY HH:MM zzz
```

where DOW is the day of the week (eg: Monday), "MM" is the day of the month, "YYYY" is the year, "HH:MM" is the time, and "zzz" is the time zone. For example:

```
Date: Thursday, 9 April 1982 19:43 est
```

#### From:

specifies the authors of the message. Its format is:

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

From: address-list

where address-list is one or more addresses separated by commas. Each address in the list identifies one of the authors of the message.

Subject:

gives a brief description of the content of the message. Its format is:

Subject: STR

where STR is the text of the subject of the message.

Sender:

identifies the user who sent the message. It is present if there is more than one address in the From field, or if the single address in the From field does not identify the user who actually sent the message (e.g., a secretary sending mail on behalf of a manager). Its format is:

Sender: address

Reply-To:

specifies the recipient(s) of any reply to this message. If this field is not present, the reply is sent to the authors of the message identified in the From field. Its format is:

Reply-To: address-list

To:

specifies the primary recipients of this message. Its format is:

To: address-list

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

where each address in the list identifies one of the primary recipients of the message.

**cc:**  
specifies the secondary recipients of the message. Its format is:

cc: address-list
------------------

where each address in the list identifies one of the secondary recipients of the message.

**bcc:**  
identifies the tertiary recipients of the message (i.e., those who receive a "blind" copy). Its format is:

bcc: address
--------------

where address identifies the tertiary recipient who received this copy of the message. The copy of a message delivered to the primary and secondary recipients never includes a bcc field.

**Acknowledge-To:**  
identifies the user to whom acknowledgements of the receipt of this message are to be sent. This field is only present in copies of the message which have not yet been acknowledged. Its format is:

Acknowledge-To: address
-------------------------

**In-Reply-To:**  
identifies the message(s) to which this message is a reply. Its format is:

In-Reply-To: STR1, STR2, ... STRn
-----------------------------------

where each STR<sub>i</sub> identifies one of the messages for which this message is a reply. The format of STR looks like this:



---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

Message of 18 June 1982 12:23 est from Spry.Proj

where "Spry.Proj" identifies the author of the original message, and the rest of the line identifies the date and time when the original message was created.

**Message-ID:**

uniquely identifies this message. Its format is:

Message-ID: <YYMMDDHHMMSS.FFFFFFFF>

where "YYMMDDHHMMSS.FFFFFFFF" is the request ID representing the time when this message was first created. For a description of request IDs, see the Reference manual.

There are two groups of header fields that appear optionally. When present, the Forwardings and comments header fields appear after the above standard fields and any non-standard fields in the header. These groups may be present in the header more than once; each occurrence of such a group identifies a single forwarding or commenting of the message.

The group of fields containing forwarding information indicates that this message was redistributed (forwarded) by one of its recipients to one or more additional recipients. If present, the Comment field contains any comments the recipient added at the time they forwarded the message. The format of this group is:

Redistributed-Date: DD Month YYYY HH:MM zzz  
Redistributed-By: address  
Redistributed-To: address-list  
Comment: STR

where "DD Month YYYY HH:MM zzz" indicates the date and time when the message was forwarded, address identifies the individual who forwarded the message, and the addresses in the address-list indicate to whom the message was forwarded.

The Comment fields indicates that a comment was added to this message. The format of this group is:

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

```
Comment-Date: DD Month YYYY HH:MM zzz
Comment-By: address
Comment: STR
```

where "DD Month YYYY HH:MM zzz" specifies the date and time that the comment was added, address identifies the user who added the comment, and STR is the actual text of the comment.

### REQUESTS

In the following send\_mail request descriptions, "spec" means "message\_specifier", "-ca" means "-control\_args", and "-selca" means "-selection\_args". See the read\_mail description for information on message specifiers and selection arguments.

?

prints a multi-columnar list of the send\_mail requests.

.

prints a line identifying the current version of send\_mail and the current state of the message being created:

```
send_mail 6.0: 23 lines (modified) Subject: Zoots
```

The word "modified" indicates that the message has been changed since the last use of the send request. The string "send\_mail 6.0" gives the version number of send\_mail. If the current recursion level is greater than one, it is included in parentheses, for example:

```
send_mail 6.0 (level 2): 5 lines:
```

If abbrev expansion is enabled, the word "abbrev" is included in parentheses before the recursion level (if any):

```
send_mail 6.0 (abbrev) (level 2): 5 lines:
```

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

.. STR

passes a command line, specified by STR, directly to the command processor, without processing by the send\_mail request processor. The ".." string must be the first two characters of the request line.

abbrev {-ca}

ab {-ca}

controls abbreviation processing within send\_mail. If invoked with no arguments, this request enables abbrev processing within send\_mail using the profile that was last used in this send\_mail invocation. If abbrev processing was not previously enabled, the profile in use at Multics command level is used; this profile is normally [home\_dir]>Person.id.profile. (See the Commands manual for a description of abbreviation processing.) Control arguments may be chosen from the following:

-off

specifies that abbreviations are not to be expanded.

-on

specifies that abbreviations are expanded. This is the default.

-profile path

specifies that the segment named by path is to be used as the profile segment. The suffix ".profile" is added to path if it is not present. The segment named by path must exist prior to the use of this control argument.

[abbrev]

returns "true" if abbreviation expansion of request lines is currently enabled within send\_mail, and "false" otherwise.

answer STR {-ca} request\_line

provides preset answers to questions asked by another request. The arguments are:

STR

is the desired answer to any question. If the answer is more than one word, it must be enclosed in quotes. If STR is -query, the question is passed on to the user. The -query control argument is the only one that can be used in place of STR.

request\_line

is any send\_mail request line. It can contain any number of separate arguments (i.e., have spaces within it) and need not be enclosed in quotes.

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

Control arguments may be chosen from the following:

-brief

-bf

suppresses printing (on the user's terminal) of both the question and the answer.

-call STR

evaluates the active string STR to obtain the next answer in a sequence. The active string is constructed from send\_mail active requests and Multics active strings (using send\_mail's "execute" active request). The outermost level of brackets must be omitted and the entire string must be enclosed in quotes if it contains request processor special characters. The return value "true" is translated to "yes", and "false" to "no". All other return values are passed as is.

-exclude STR

-ex STR

passes on, to the user or other handler, questions whose text matches STR. If STR is surrounded by slashes (/), it is interpreted as a qedx regular expression. Otherwise, answer tests whether STR is literally contained in the text of the question. Multiple occurrences of -match and -exclude are allowed (see "Notes" below). They apply to the entire request line.

-match STR

answers only questions whose text matches STR. If STR is surrounded by slashes (/), it is interpreted as a qedx regular expression. Otherwise, answer tests whether STR is literally contained in the text of the question. Multiple occurrences of -match and -exclude are allowed (see "Notes" below). They apply to the entire request line.

-query

skips the next answer in a sequence, passing the question on to the user. The answer is read from the user\_i/o I/O switch.

-then STR

supplies the next answer in a sequence.

-times N

gives the previous answer (STR, -then STR, or -query) N times only (where N is an integer).

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

Answer provides preset responses to questions by establishing an on unit for the condition command\_question, and then executing the designated request line. If any request in the request line calls the command\_query\_ subroutine (described in the Subroutines manual) to ask a question, the on unit is invoked to supply the answer. The on unit is reverted when the answer request returns to send\_mail request level. See the Reference manual for a discussion of the command\_question condition.

If a question is asked that requires a yes or no answer, and the preset answer is neither "yes" nor "no", the on unit is not invoked.

The last answer specified is issued as many times as necessary, unless followed by the -times N control argument.

The -match and -exclude control arguments are applied in the order specified. Each -match causes a given question to be answered if it matches STR; each -exclude causes it to be passed on if it matches STR. A question that has been excluded by the -exclude control argument is reconsidered if it matches a -match later in the request line.

append path

app path

appends the message (with header) to the end of the ASCII segment specified by path. The suffix .mail is added to path if it is not present. If the specified segment does not already exist, the user is asked whether to create it.

apply {-ca} STR

ap {-ca} STR

places the message in a temporary segment in the process directory, then concatenates the command line specified by STR with intervening spaces and appends the pathname of the temporary segment. This concatenated command line is passed to the Multics command processor. When the command line has completed, the message in send\_mail is replaced with the contents of the temporary segment. This request can be used to edit the message with a text editor. Control arguments are:

-fill

-fi

specifies that the message text is reformatted after the command line has been executed.

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

-header

-he

specifies that the message header is passed to the command line in addition to the message text.

-line\_length N

-ll N

specifies the line length to use when reformatting the message text. If this control argument is not given, the line length specified on the send\_mail command line is used. If no line length is specified on the send\_mail command line, a line length of 72 is used.

-no\_fill

-nfi

specifies that the message text is not be reformatted.

-no\_header

-nhe

specifies that only the message text is passed to the command line. This is the default.

The supplied command line for the apply request need not be enclosed in quotes. However, if there are (), [], or "'s in the command line that should be processed by the Multics command processor, they should be enclosed in quotes to prevent processing by send\_mail's request processor.

The message is passed to the Multics command line by placing the message text and header (if requested) into a temporary segment. The pathname of this segment is appended to the command line, which is then executed. The contents of the segment after execution replace the prior message text (and header).

This request may be used to edit the message with an editor other than qedx. For example, the following request invokes the Emacs text editor on the message text:

```
apply emacs
```

The default for reformatting the message after execution of the command line is dependent on the original source of the message text. If terminal input was used, the default is to reformat the message; if file input was used, the default is to leave the message unformatted. This default may be changed by use of the -fill and -no\_fill control arguments on the send\_mail command line. Additionally, whatever default is specified may be overridden for one invocation of the apply request by use of the control arguments described above.

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

If the `-header` control argument is specified, both the message header and text are placed in the temporary segment.

After apply execution is complete, `send_mail` analyzes the new message and then updates the message's subject, In-Reply-To field, lists of primary/secondary recipients, authors, and list of recipients for future replies.

`cc {addresses}`  
adds any addresses specified to the list of secondary recipients of the message. Mail is sent to these addresses when a subsequent send request is issued with no arguments. The addresses are added to the `cc` field, which is created if necessary. If no addresses are specified, the secondary recipients of the message are listed.

`copy path`  
`cp path`  
copies the message into the mailbox designated by path. The mailbox must already exist. The `.mbx` suffix is added to path if it is not present.

`do STR {args}`  
or  
`do {-ca}`  
expands a request line specified by `STR` by substituting the supplied arguments into the line before execution. Arguments are character string arguments that replace parameters in the request line.

The following control arguments set the mode of operation of the `do` request:

`-absentee`  
an `any_other` handler is established that catches all conditions and aborts execution of the request line without aborting the process.

`-brief`  
`-bf`  
the expanded request line is not printed before execution. This is the default.

`-go`  
the expanded request line is passed on for execution. This is the default.

`-interactive`  
the `any_other` handler is not established. This is the default.

-long

-lg

the expanded request line is printed before execution.

-nogo

the expanded request line is not passed on for execution.

Any sequence beginning with & in the request line is expanded by the do request using the arguments given on the request line. Following is the list of parameters:

&I

is replaced by argI. I must be a digit from 1 to 9.

&(I)

is also replaced by argI. I can be any value, however.

&qI

is replaced by argI with any quotes in argI doubled. I must be a digit from 1 to 9.

&q(I)

is also replaced by argI with any quotes doubled. I can be any value.

&rI

is replaced by all the arguments starting with argI. Each argument is placed in quotes with contained quotes doubled. I must be a digit from 1 to 9.

&r(I)

is also replaced by a requoted argI. I can be any value.

&fI

is replaced by all the arguments starting with argI. I must be a digit from 1 to 9.

&f(I)

is also replaced by all the arguments starting with argI. I can be any value.

&qfI

is replaced by all the arguments starting with argI with any quotes doubled. I must be a digit from 1 to 9.

&qf(I)

is also replaced by all the arguments starting with argI with quotes doubled. I can be any value.



---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

&rf(I)

is also replaced by all the arguments starting with argI, requoted. I can be any value.

&&

is replaced by an ampersand.

&!

is replaced by a 15 character unique string. The string used is the same everywhere &! appears in the request line.

&n

is replaced by the actual number of arguments supplied.

&f&n

is replaced by the last argument supplied.

[do request\_line {args}]

returns a request line with argument substitution.

exec\_com path {args}

ec path {args}

executes a program written in the exec\_com language, where path is the pathname of an exec\_com program. The suffix "sdmec" is added to the pathname if necessary. This program is used to pass request lines to send\_mail and to pass input lines to requests which read input. The arguments are optional arguments to the exec\_com program and are substituted for parameter references in the program such as &1.

If the pathname does not contain a "<" or ">" character, send\_mail searches for the exec\_com program using the mail\_system search list. The default content of this search list is:

<pre>-working_dir &gt;udd&gt;[user project]&gt;[user name]&gt;[user name].mlsys</pre>
---

When evaluating a send\_mail exec\_com program, subsystem active requests are used rather than Multics active functions when evaluating the &[...] construct and the active string in an &if statement. The send\_mail execute active request may be used to evaluate Multics active strings within the exec\_com.

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

Currently, any error detected during execution of an `exec_com` within `send_mail` aborts the request line in which the `exec_com` request was invoked.

[`exec_com path {args}`]

[`ec path {args}`]

executes a program written in `exec_com` language that specifies a return value of the `exec_com` request with the `&return` statement. The arguments are the same as for the `exec_com` request.

execute STR

e STR

executes the supplied line as a Multics command line, where STR is the Multics command line to be executed or the Multics active string to be evaluated. It need not be enclosed in quotes.

The recommended method to execute a Multics command line from within `send_mail` is the `".."` escape sequence. The `execute` request is intended as a means of passing information from `send_mail` to the Multics command processor.

All `()`, `[]`, and `'`'s in the given line are processed by the `send_mail` request processor, not the Multics command processor. Thus, the values of subsystem active requests may be passed to Multics commands when using the `execute` request. For example, the `send_mail` request line:

```
e sm Roe.NewProj I'm sending you mail about [subject].
```

warns user `Roe.NewProj` that she is about to receive a message.

[`execute STR`]

[`e STR`]

the `execute` active request can be used with a Multics active function to invoke the active function from within `send_mail`. For example, the following `send_mail` request line:

```
write [e date]
```

writes the ASCII representation of the message being created into a segment in the working directory. The entry name of

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

this segment is the current date with a suffix of ".mail" (e.g., 12/01/82.mail).

fill {-ca}

fi {-ca}

reformats the message text according to "fill-on" and "align-left" modes of the compose command. If the -fill control argument, which is the default for terminal input, is specified on the send\_mail command line, the message is reformatted after each use of the qedx and apply requests. This automatic reformatting can be overridden by use of the -no\_fill control argument to these requests. The control argument to the fill request is:

-line\_length N

-ll N

specifies the maximum line length. The default is 72 characters, or the value specified with the send\_mail -line\_length control argument.

from {addresses}

adds addresses to the list of authors of the message if any addresses are specified. The addresses are added to the From field of the header. If no addresses are specified, the authors of the message are listed. If no explicit authors are specified, either via this request or via use of the -from control argument on the send\_mail command line, the user of send\_mail is listed as the sole author of the message when it is transmitted. If a message has more than one author or the author is not the user using send\_mail, a Sender field identifying the user of send\_mail is added to the message when it is transmitted.

help {STR}

prints information about various send\_mail topics, including detailed descriptions of send\_mail requests. If specified, STR is the name of a topic on which information is to be printed. If STR is not specified, the help request lists the requests that provide information about send\_mail.

The help request accepts most of the control arguments accepted by the Multics help command. Type ".. help help" for a complete description of the help request. Following is a description of some of the more useful control arguments for the help request:

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

-brief

-bf

prints only a summary of a request or active request, including the Syntax section, list of arguments, control arguments, etc.

-search STRs

-srh STRs

begins printing with the paragraph containing all the strings STRs. By default, printing starts at the beginning of the information.

-section STRs

-scn STRs

begins printing at the section whose title contains all the strings STRs. By default, printing starts at the beginning of the information.

-title

prints section titles and section line counts; then asks if the user wants to see the first paragraph of information.

The most useful responses to questions asked by the help request are:

?

prints the list of responses allowed to help queries.

.

prints "help" to identify the current interactive environment.

.. command\_line

treats the remainder of the response as a Multics command line.

no

n

stops printing information for this topic and proceeds to the next topic, if any.

quit

q

stops printing information for this topic and returns to the subsystem's request level.

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

rest {-scn}  
r {-scn}  
prints remaining information for this topic without intervening questions. If -section or -scn is given, help prints only the rest of the current section without questions and then asks if the user wants to see the next section.

search {STRs} {-top}  
srh {STRs} {-top}  
skips to the next paragraph containing all the strings STRs. If -top or -t is given, searching starts at the top of the information. If STRs are omitted, help uses the STRs from the previous search response or the -search control argument.

section {STRs} {-top}  
scn {STRs} {-top}  
skips to the next section whose title contains all the strings STRs. If -top or -t is given, title searching starts at the top of the information. If STRs are omitted, help uses the STRs from the previous section response or the -section control argument.

skip {-scn} {-seen}  
s {-scn} {-seen}  
skips to the next paragraph. If -section or -scn is given, help skips all paragraphs of the current section. If -seen is given, help skips to the next paragraph which the user has not seen. Only one control argument is allowed in each skip response.

title {-top}  
lists titles and line counts of the sections that follow; if -top or -t is given, help lists all section titles. The previous question is repeated after titles are printed.

yes  
y  
prints the next paragraph of information on this topic.

if [EXPR] -then LINE1 {-else LINE2}  
conditionally executes one of two request lines depending on the value of an active string. The arguments are:

EXPR  
is the active string which must evaluate to either "true" or "false". The active string is constructed from

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

send\_mail active requests and Multics active strings (using send\_mail's execute active request).

LINE1

is the send\_mail request line to execute if EXPR evaluates to "true". If the request line contains any request processor characters, it must be enclosed in quotes.

LINE2

is the send\_mail request line to execute if EXPR evaluates to "false". If omitted and EXPR is "false", no additional request line is executed. If the request line contains any request processor characters, it must be enclosed in quotes.

[if [EXPR] -then STR1 {-else STR2}]

returns one of two character strings to the send\_mail request processor, depending on the value of an active string. The arguments are:

EXPR

is the active string that must evaluate to either "true" or "false". The active string is constructed from send\_mail active requests and Multics active strings (using send\_mail's execute active request).

STR1

is returned as the value of the if active request if the EXPR evaluates to "true".

STR2

is returned as the value of the if active request if the EXPR evaluates to "false". If omitted and the EXPR is "false", a null string is returned.

in\_reply\_to {STRs}

irt {STRs}

replaces the In-Reply-To field of the message (if any) with the concatenation of the STRs with intervening spaces. If no STRs are specified, it prints the contents of the In-Reply-To field.

list\_help {topics}

lh {topics}

displays the name of all send\_mail information segments on given topics. If no topics are given, all send\_mail information segments are listed.

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

When matching topics with info segment names, an info segment name is considered to match a topic only if that topic is at the beginning or end of a word within the segment name. Words in info segment names are bounded by the beginning and end of the segment name and by the characters period (.), hyphen (-), underscore (\_), and dollar sign (\$). The ".info" suffix is not considered when matching topics.

list\_original {spec} {-selca} {-ca}

lso {spec} {-selca} {-ca}

provides a one-line summary of relevant information about the message(s) being answered. This request is only available within a send\_mail that was created by the read\_mail reply request. It accepts read\_mail message specifiers, so that the user can examine other messages which might be relevant to the reply. Control arguments are:

-header

-he

precedes the message listing by a header line that identifies the columns of the list. This is the default.

-include\_deleted

-idl

includes all messages in the mailbox, whether or not they have been deleted, when processing message\_specifiers and selection\_args to determine which messages will be listed.

-line\_length N

-ll N

uses the supplied line length when determining where and if to truncate the message subject. The default length is the terminal's line length. default.

-no\_header

-nhe

omits the header line from the listing.

-no\_line\_length

-nll

does not truncate the message subject unless the subject is more than one line long.

-no\_reverse

-nrv

lists the messages in ascending numeric order. This is the default.

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

-only\_deleted

-odl

includes only those messages which have been deleted.

-only\_non\_deleted

-ondl

includes only those messages which have not been deleted.  
This is the default.

-reverse

-rv

lists the messages in descending numeric order.

If this request was created by the reply request in read\_mail, you can list any message in the read\_mail invocation with this request.

[list\_original {spec} {-selca} {-ca}]

[lso {spec} {-selca} {-ca}]

returns the message numbers of the messages being answered by send\_mail. This active request is only available within an invocation of send\_mail that was created by the read\_mail reply request. It takes the same control arguments as the list\_original request.

list\_requests {STR} {-ca}

lr {STR} {-ca}

prints a brief description of selected send mail requests, where STR specifies the request(s) to be described. Any request with a name containing one of these strings is listed unless -exact is used, in which case the request name must exactly match one of these strings. When matching STRs with request names, a request name is considered to match a STR only if that STR is at the beginning or end of a word within the request name. Words in request names are bounded by the beginning and end of the request name and by the characters period (.), hyphen (-), underscore (\_), and dollar sign (\$).

Control arguments are:

-all

-a

includes undocumented and unimplemented requests in the list of requests eligible for matching the STR arguments.

-exact

lists only those requests one of whose names exactly match one of the STR arguments.



---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

log

saves a copy of the message in the user's logbox (Person\_id.sv.mbx). This request creates the logbox if one does not already exist.

log\_original {spec} {-ca}

logo {spec} {-ca}

places a copy of the original message(s) into the user's logbox. This request is only available within a send\_mail that was created by the read\_mail reply request. It accepts read\_mail message specifiers, so that the user can log other messages which might be relevant to the reply.

The user's logbox is the mailbox >udd>Project\_id>Person\_id>Person\_id.sv.mbx. This mailbox is created automatically by the request if it does not already exist. The user is informed when the logbox is created. This request acknowledges any messages requiring acknowledgement unless -no\_acknowledge is specified on the read\_mail command line.

Control arguments are:

-include\_deleted

-idl

includes all messages in the mailbox, whether or not they have been deleted, when processing the message\_specifiers to determine which messages will be logged.

-only\_deleted

-odl

includes only those messages which have been deleted.

-only\_non\_deleted

-ondl

includes only those messages which have not been deleted. This is the default.

-no\_reverse

-nrv

logs the messages in ascending numeric order. This is the default.

-reverse

-rv

logs the messages in descending numeric order.

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

message\_id

mid

prints the Message-ID field of this message, creating the field if necessary.

preface path

prf path

same as the append request, but inserts the message at the beginning of the ASCII segment specified by path.

print {-ca}

pr {-ca}

p {-ca}

prints the message. The control argument may be one of the following:

-brief\_header

-bfhe

prints an abbreviated form of the message header, including the Subject and To fields. If the message has no subject, the Subject line is omitted. If there are no primary recipients for the message, the To line contains the string <no addresses>. If there is no secondary recipient, the cc line is omitted. This is the default.

-header

-he

prints the complete message header with the message.

-no\_header

-nhe

does not include a message header with the message text.

print\_header {-ca}

prhe {-ca}

prints the header of the message. The control argument may be one of the following:

-brief

-bf

prints an abbreviated form of the message header, including the Subject and To fields.

-long

-lg

prints the complete message header. This is the default.

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

print\_original {spec} {-selca} {-ca}

pro {spec} {-selca} {-ca}

prints the original message(s). This request is only available within a send\_mail that was created by the read\_mail reply request. It accepts read\_mail message specifiers, so that the user can examine other messages which might be relevant to the reply. This request acknowledges any messages requiring acknowledgement unless -no\_acknowledge is specified on the read\_mail command line. It takes the same control arguments as the log\_original request, and these additional control arguments:

-header

-he

prints the full header associated with each message.

-no\_header

-nhe

prints a summary of the header associated with each message.

print\_original\_header {spec} {-selca} {-ctl\_args}

prohe {spec} {-selca} {-ctl\_args}

prints message header(s) of the original message(s). This request is only available within a send\_mail that was created by the read\_mail reply request. It accepts read\_mail message specifiers, so that the user can examine other messages which might be relevant to the reply. This request acknowledges any messages requiring acknowledgement unless -no\_acknowledge is specified on the read\_mail command line. It takes the same control arguments as the log\_original request.

gedx {-ca}

qx {-ca}

invokes the gedx editor to modify the message. The gedx w (write) request is not necessary to reflect changes in the message to send\_mail. The editor request line 1,\$dr can be used to restore the original text.

The default for reformatting the message after editing is dependent on original source of the message text. If terminal input was used, the default is to reformat the message; if file input was used, the default is to leave the message unformatted. This default may be changed by use of the -fill and -no\_fill control arguments on the send\_mail command line. Additionally, whatever default is specified may be overridden for one invocation of the gedx request by use of the control arguments described below.

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

If the `-header` control argument is specified, both the message header and text are be given to the editor. After editing is complete, `send_mail` analyzes the new message and then updates the message's subject, In-Reply-To field, lists of primary/secondary recipients, authors, and list of recipients for future replies.

The control arguments are:

`-fill`

`-fi`

causes the message text to be reformatted after editing.

`-header`

`-he`

both header and text can be edited.

`-line_length N`

`-ll N`

specifies the line length of the reformatted text. If this control argument is not given, the line length (if any) specified on the `send_mail` command line is used; otherwise, a line length of 72 characters is used.

`-no_fill`

`-nfi`

specifies that the message text is not reformatted.

`-no_header`

`-nhe`

only the message text can be edited. This is the default.

`quit {-ca}`

`q {-ca}`

exits the `send_mail` command. The control argument can be one of the following:

`-force`

`-fc`

does not ask about a modified or incomplete message before returning to command level.

`-no_force`

`-nfc`

causes `send_mail` to query the user for permission to exit if the message has been modified since it was last sent, saved, or written. This is the default.

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

ready  
rdy

prints a Multics ready message. The Multics general\_ready command may be used to change the format of the ready message printed by this request, and also after execution of request lines if the ready\_on request is used. The default ready message gives the time of day, the amount of CPU time, and page faults used since the last ready message was typed.

ready\_off  
rdf

does not generate a ready message after the execution of each request line. This is the default.

ready\_on  
rdn

prints a ready message after the execution of each request line.

remove {addresses} {-ca}  
rm {addresses} {-ca}

deletes specified addresses and/or specified header fields. All occurrences of the addresses are removed from both the list of primary recipients and the list of secondary recipients. If no addresses are given, at least one of the control arguments described below must be used. New recipients, authors, etc. can be added to the message with the cc, from, in\_reply\_to, message\_id, reply\_to, subject, and to requests. Control arguments may be chosen from the following:

-all  
-a

removes all recipients from the message. This control argument must appear before all other control arguments, and may not be used if any addresses are specified.

-cc {addresses} {-ca}

deletes specified addresses from the cc field, or deletes the entire field if -all (-a) is given. Either an address or -all must be supplied.

-from {addresses} {-ca}

deletes specified addresses from the From field, or deletes the entire field if -all (-a) is given. Either an address or -all must be supplied.

-in\_reply\_to

-irt

deletes the In\_Reply\_To field.

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

-message\_id

-mid

deletes the Message\_ID field.

-reply\_to {addresses} {-ca}

deletes specified addresses from the Reply\_To field, or deletes the entire field if -all (-a) is given. Either an address or -all must be supplied.

-subject

-sj

deletes the Subject field.

-to {addresses} {-ca}

deletes specified addresses from the To field, or deletes the entire field if -all (-a) is given. Either an address or -all must be supplied.

reply\_to {addresses}

rpt {addresses}

adds addresses of users who are to receive the reply to this message. These addresses are also appended to the Reply-To field of the header, which is created if necessary. If no addresses are specified, read\_mail sends replies to this message to the authors of the message.

save path

sv path

saves a copy of the message in the indicated savebox. The suffix ".sv.mbx" is added to path if not already present. If the savebox does not exist, the user is asked whether to create it.

save\_original {spec} path {-ca}

svo {spec} path {-ca}

saves the original message(s) into a savebox. If the savebox identified by the path argument does not exist, the user is queried for permission to create it. This request is only available within a send\_mail that was created by the read\_mail reply request; any message within the read\_mail invocation may be saved by this request. Any message requiring acknowledgement is acknowledged by this request unless -no\_acknowledge is specified on the read\_mail command line. Control arguments for the save\_original request are the same as for the log\_original request.

send {addresses} {-ca}

transmits the message to the primary and secondary recipients if no addresses are specified. If any addresses are specified, the message is transmitted only to these

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

addresses, without adding them to the message header. It is possible to send "blind" carbon copies by issuing two separate send requests; one without addresses to deliver the message to the primary and secondary recipients, and a second to deliver the message to the blind carbon recipients.

The following send request control arguments are identical to the send\_mail command control arguments of the same name:

-abort	-no_abort
-acknowledge (-ack)	-no_acknowledge (-nack)
-brief (-bf)	-no_header (-nhe)
-header (-he)	-no_message_id (-nmid)
-long (-lg)	-save_path (-sv path)
-message_id (-mid)	

The above control arguments temporarily override the defaults specified on the send\_mail command line.

subject {STRs}

sj {STRs}

replaces the Subject field of the message (if any) with the concatenation of the STRs with intervening spaces. If no STRs are specified, the contents of the Subject field are printed instead.

[subject]

[sj]

returns the contents of the Subject field as a single quoted string.

subsystem\_name

prints the name of the current subsystem.

[subsystem\_name]

returns the name of the current subsystem. This active request is useful as part of an abbrev which is shared by multiple subsystems.

subsystem\_version

prints the version of the current subsystem.

[subsystem\_version]

returns the version of the current subsystem. This active request may be used in an abbrev which is shared by multiple subsystems.

to {addresses}

adds addresses to the list of primary recipients of the message or prints the contents of the list. When a

---

send\_mail (sdm)

---

---

send\_mail (sdm)

---

subsequent send request is issued with no arguments, mail is sent to the addresses in the primary and secondary recipient lists. The addresses are added to the To field of the header, which is created if necessary. If no addresses are specified, the primary recipients of the message are listed.

write\_path {-ca}

appends the message (with header) to the ASCII segment designated by path. The suffix .mail is added to path if it is not present. The segment is created if necessary. The control argument may be one of the following:

-extend

-ex

appends the message to the end of the segment. This is the default.

-truncate

-tc

truncates the segment before writing the message to it.

write\_original {spec} path {-ca}

wo {spec} path {-ca}

writes the original message(s) into an ASCII segment specified by path. This request is only available within a send\_mail that was created by the read\_mail reply request; any message within the read\_mail invocation may be written by this request. Any message requiring acknowledgement is acknowledged by this request unless -no\_acknowledge is specified on the read\_mail command line. The write\_original request takes the same control arguments as the log\_original request, and the following additional control arguments:

-extend

writes the messages at the end of the segment if there is already data present in the segment. This is the default.

-truncate

-tc

truncates the segment before writing the messages.



## APPENDIX B

### MAILBOX COMMANDS

The mailbox access commands and the `mbx_create` command are documented in this appendix. Extended access provides a way to further your control over your mailboxes.

The extended access modes for mailboxes are:

add (a)	add a message
delete (d)	delete any message
read (r)	read any message
own (o)	read or delete only your own messages; that is, those sent by you
status (s)	find out how many messages are in the mailbox
wakeup (w)	can send a wakeup indicating that a message was added to the mailbox

The extended access placed on a new mailbox is:

adros	user who created the mailbox
aw	*.SysDaemon.*
aw	*.*.*

Users have full (adros) access to their personal mailbox (`Person_id.mbx`).

When assigning or removing access to your mailboxes for other users, `User_ids` are used. The matching strategy for access control names is as follows:

1. A literal component name, including "\*", matches only a component of the same name.

2. A missing component name not delimited by a period is taken to be a literal "\*" e.g., "\*.Multics" is treated as "\*.Multics.\*"). Missing components on the left must be delimited by periods.
3. A missing component name delimited by a period matches any component name.

Some examples of access names and which ACL entries they match are:

*.*.*	matches only the ACL entry "*.*.*".
Multics	matches only the ACL entry "Multics.*.*". absence of a leading period makes Multics the first component.
.Multics	matches every ACL entry with middle component of Multics.
..	matches every ACL entry.
.	matches every ACL entry with a last component of "*".
""	(null string) matches every entry ending in ".*.*".

---

mbx\_create (mbr)

---

---

mbx\_create (mbr)

---

mbx\_create (mbr)

The `mbx_create` command creates a mailbox with a specified name in a specified directory.

#### SYNTAX AS A COMMAND

`mbx_create paths`

#### ARGUMENTS

`paths`

are the pathnames of mailboxes to be created. If `pathi` does not have the `.mbx` suffix, one is assumed.

#### NOTES

The user must have modify and append permission on the directory in which he is creating a mailbox.

If the creation of a mailbox introduces a duplication of names within the directory, and if the old mailbox has only one name, the user is asked for permission to delete the old mailbox. If the answer is "no", no action is taken. If the old mailbox has multiple names, the conflicting name is removed and a message to that effect is issued to the user.

See also the `mbx_set_acl` command in this appendix.

#### EXAMPLES

The command line:

```
! mbr Green Hogan.home >udd>Multics>Gillis>Gillis
```

creates the mailboxes `Green.mbx` and `Hogan.home.mbx` in the working directory and creates the mailbox `Gillis.mbx` in the directory `>udd>Multics>Gillis`.

---

mbx\_delete\_acl (mbda)

---

---

mbx\_delete\_acl (mbda)

---

mbx\_delete\_acl (mbda)

The `mbx_delete_acl` command deletes entries from the access control list (ACL) of a given mailbox.

#### SYNTAX AS A COMMAND

`mbx_delete_acl path {User_ids} {-control_args}`

#### ARGUMENTS

`path`  
is the pathname of a mailbox. The `.mbx` suffix is assumed if not supplied. The star convention is allowed.

`User_ids`  
are access control names of the form `Person_id.Project_id.tag`. All entries with matching names are deleted. If no `User_ids` are given, the user's own is assumed.

#### CONTROL ARGUMENTS

`-all`  
`-a`  
deletes all entries except for `*.*.*`.

`-brief`  
`-bf`  
suppresses the messages "User name not on ACL" and "Empty ACL".

`-chase`  
chases links when using the star convention.

`-no_chase`  
does not chase links when using the star convention. This is the default.

#### NOTES

The user must have modify permission on the containing directory.

See the beginning of this appendix for an explanation of `User_id` matching strategy.

---

mbx\_list\_acl (mbla)

---

---

mbx\_list\_acl (mbla)

---

mbx\_list\_acl (mbla)

The mbx\_list\_acl command lists entries on the access control lists of mailboxes.

#### SYNTAX AS A COMMAND

mbx\_list\_acl path {User\_ids} {-control\_args}

#### ARGUMENTS

path

is the pathname of a mailbox. The .mbx suffix is assumed if not supplied.

User\_ids

are access control names of the form Person\_id.Project\_id.tag. All entries with matching names are listed. If no User\_ids are given, the entire ACL is listed.

#### CONTROL ARGUMENTS

-brief

-bf

suppresses the message "User name not on ACL".

-chase

chases links matching a starname. The default is to chase a link only when specified by a non-starred pathname.

-no\_chase

does not chase links when using the star convention. This is the default.

#### NOTES

Status permission is required on the parent directory.

The active function has the following syntax:

[mbla path {User\_ids}]

---

mbx\_list\_acl (mbla)

---

---

mbx\_list\_acl (mbla)

---

| It returns the modes and access names of matching entries  
| separated by spaces (e.g., "adrosw A.B.\* ao C.D.a"). The -brief  
| control argument is assumed.

---

mbx\_set\_acl (mbsa)

---

---

mbx\_set\_acl (mbsa)

---

mbx\_set\_acl (mbsa)

The `mbx_set_acl` command manipulates the access control lists of mailboxes.

#### SYNTAX AS A COMMAND

`mbx_set_acl path mode1 User_id1 ... modeN {User_idN} {-ctl_args}`

#### ARGUMENTS

`path`

is the pathname of a mailbox. The `.mbx` suffix is assumed if not supplied. The star convention is allowed.

`modeN`

is an extended access mode, consisting of any or all of the letters "adrosw" or "null", "n", or "" for null access.

`User_idN`

are access control names of the form `Person_id.Project_id.tag`. All ACL entries with matching names are assigned `modeN`. If no match is found and all three components are given, an entry for `User_idN` is added to the ACL. If the last `User_id` is omitted, the user's `Person_id` and `Project_id` are assumed.

#### CONTROL ARGUMENTS

`-brief`

`-bf`

suppresses the message "No match for `User_id`" on ACL of `<path>`, where `User_id` omits components.

`-chase`

chases links matching a starname. Links are always chased when `path` is not a starname.

`-no_chase`

does not chase links when using the star convention. This is the default.

`-no_sysdaemon`

`-nsd`

suppresses the addition of an "aow `*.SysDaemon.*`" term when using `-replace`.

---

mbx\_set\_acl (mbsa)

---

---

mbx\_set\_acl (mbsa)

---

-replace

-rp

deletes all ACL terms (with the exception of the default \*.SysDaemon.\* term unless -no\_sysdaemon is specified) before adding the terms specified on the command line. The default is to add to and modify the existing ACL.

-sysdaemon

-sd

with -replace, adds an "aow \*.SysDaemon.\*" ACL term before adding the terms specified on the command line.

### NOTES

The user must have modify permission on the containing directory.

See the beginning of this appendix for an explanation of User\_id matching strategy.



## APPENDIX C

### GLOSSARY

The following list of terms is a supplement to the glossary provided in the New Users' Intro - Part I. Most of the terms appear for the first time here, but several are repeated.

#### ADDRESS

a form of name that directs mail system commands to mailboxes. The name is usually an entryname (Ching.mbx), a full pathname (>udd>ProjCat>Ching.mbx), or a User\_id (Ching.ProjCat).

#### HEADER

the group of lines preceding the text of a message, and containing information about the creation and destination of the message. Standard information included in the header is the User\_id of the person who wrote the message, the date and time it was sent, the subject of the message, and who it was sent to.

#### HEADER FIELD

one specific kind of information contained in the header, such as the message subject (the Subject field) or the lists of recipients (the To and cc fields). Information for standard header fields is usually supplied automatically, but most header fields can be controlled with send\_mail requests.

\*

## LOGBOX

a mailbox in the home directory to which only the owner has access. It is created with the `log` request or the `send_mail -log` control argument, and has the pathname `>udd>Project_id>Person_id>Person_id.sv.mbx`. The logbox is intended as a general mail storage container; see also `savebox`.

## MAILBOX

a container for mail system messages, controlled by a set of extended access modes. Typically, each person has a mailbox named `Person_id.mbx` under the home directory, to which senders have limited access (access to read and delete the messages they send). Users may also create other mailboxes, called logboxes and saveboxes.

## MESSAGE

in this manual, a "message" refers to a mail system message created by a user with the `send_mail` command. Other types of message are referred to by more specific names, such as interactive messages and error messages.

## MESSAGE SPECIFIER

a combination of message numbers, keywords, character strings, and logical and arithmetic operators that are used with various `read_mail` requests to specify which messages are to be manipulated.

## REQUEST LINE

one complete instruction, within the request loop, to `send_mail` or `read_mail`. It includes the request name, any arguments to the request (such as message specifiers and request control arguments), and a newline. A request line is parallel to a Multics command line, except that a request line is issued at request level.

## REQUEST LOOP

a repeating cycle within `read_mail` and `send_mail` that prompts you for a request (e.g., `read_mail:`), reads the request you type, performs the specified operation, and finishes with a prompt to you for another request. The request loop is parallel to Multics command level, except that at command level no prompt is given.

## SAVEBOX

a mailbox created by the save request or the send\_mail -save control argument, in any directory to which the owner has access. By default, users have access only to their own saveboxes. Users can create as many saveboxes as desired, to store mail by topic.

## INDEX

### MISCELLANEOUS

! (exclamation point) 4-10  
# character 3-2  
\* (asterisk) 4-4  
. request  
  read\_mail 7-6, A-18  
  send\_mail 7-6, A-57  
.. escape request 7-5  
  re-entering mail system 7-6  
  read\_mail A-18  
  send\_mail A-58  
? request  
  read\_mail 4-12, A-18  
  send\_mail A-57  
@ character 3-2  
\f  
  send\_mail 3-5  
\g  
  read\_mail 4-7  
  send\_mail 3-3

### A

abbrev  
  active request  
    read\_mail A-19, A-58  
  request  
    profile 7-1  
    read\_mail 7-1, A-19  
    send\_mail 7-1, A-58  
accept\_messages (am) command  
  1-2, 1-6  
access 1-2  
  to mail segments 6-5  
  to mailboxes B-1  
  to sent messages 6-5  
acknowledgement 3-7, 3-14  
  Acknowledge-To field 3-7  
active functions 7-5, 7-7  
active requests 7-7  
address 3-1, 5-3, 5-4, A-52  
  -comment 5-7  
  -user and -mbx 6-5  
  deletion 5-5  
all  
  active request  
    read\_mail A-20  
  keyword 4-4, 4-5  
  request  
    read\_mail A-19

answer request  
  read\_mail A-20  
  send\_mail A-58

append request  
  read\_mail 6-6, A-22  
  send\_mail 6-6

apply request  
  read\_mail A-23  
  send\_mail 7-9, A-60

asterisk (\*) 4-4

### C

cc  
  field 5-5  
  and your User\_id 6-2, 6-4  
  request  
  send\_mail 5-4, A-62

command level 1-5  
  and request level 1-5  
  within mail system 7-5, 7-8

comments (to addresses) 5-8

control arguments 1-5, 7-3  
  addresses  
    -mbx 6-5  
  and requests 1-5  
  for addresses  
    -comment 5-7  
    -user 6-5  
  print\_mail  
    -list 2-3  
    complete list A-2  
  read\_mail 4-15  
    -list 4-16  
    -log 6-2  
    -no\_header 4-16  
    -request 7-4  
  send\_mail 3-12  
    -acknowledge 3-14  
    -input\_file 3-13  
    -log 6-2  
    -request\_loop 3-13  
    -save 6-3  
  start\_up.ec 7-4

copy request  
  read\_mail A-24  
  send\_mail A-62

current  
  active request  
    read\_mail A-25  
  keyword 4-4  
  message 4-5  
  request  
    read\_mail A-25

### D

Date field 1-3

delete request  
  read\_mail 4-3, 4-8, A-25

deletion  
  addresses 5-5  
  header fields 5-5  
  line 3-2  
  message 4-8  
  word 3-2

do  
  active request  
    read\_mail A-27  
    send\_mail A-64  
  request  
    read\_mail A-25  
    send\_mail A-62  
dprint command 6-5

### E

editor 3-4  
  other editors 7-9  
  qedx 3-4

Emacs 7-9

enter\_output\_request command  
  7-8

erase (@) character 3-2

examining mail  
  logbox 6-2  
  other mailboxes 6-5, 7-6  
  saveboxes 6-5

exclamation point (!) 4-10

execute  
  active request  
    read\_mail 7-7, 7-8, A-29  
    send\_mail 7-7, 7-8, A-65  
  request  
    read\_mail 7-8, A-28  
    send\_mail 7-8, A-65

exec\_com 1-6  
  active request  
    read\_mail A-28  
    send\_mail A-65  
  and apply request 7-9  
  request  
    read\_mail 7-9, A-27  
    send\_mail 7-9, A-64  
  start\_up 7-4

extended access 1-2, B-1

## F

fields  
  see header fields, 5-2

fill request  
  send\_mail 3-14, A-66

first  
  active request  
    read\_mail A-29  
  keyword 4-4  
  request  
    read\_mail A-29

forward request  
  read\_mail 4-8, A-29

from  
  field 1-3, 5-7  
  request  
    send\_mail 5-7, A-66

## H

header 1-3, 4-7, 5-1, A-53  
  comments 5-8  
  fields 1-3  
    Acknowledge-To 3-7  
    cc 5-5, 6-2, 6-4  
    complete list A-53  
    Date 1-3  
    From 1-3, 5-7  
    In-Reply-To 4-8  
    Redistributed-By 4-8  
    Redistributed-Date 4-8  
    Redistributed-To 4-8  
    Sender 5-7  
    Subject 1-3, 5-6, 7-7  
    To 1-3, 5-2  
  modifications 5-5

help request  
  read\_mail 4-14, A-30  
  send\_mail 3-11, A-66

## I

if  
  active request  
    read\_mail A-33  
    send\_mail A-69  
  request  
    read\_mail A-32  
    send\_mail A-68

In-Reply-To field 4-8

interactive message 1-2, 1-6

in\_reply\_to request  
  send\_mail A-69

## K

keywords 4-4

## L

last  
 active request  
 read\_mail A-33  
 keyword 4-4  
 request  
 read\_mail A-33

link command 1-2

linking mailboxes 1-2

list  
 active request  
 read\_mail A-36  
 request  
 read\_mail 4-2, 4-3, 4-8,  
 A-34  
 -idl 4-10

list\_help request  
 read\_mail A-36  
 send\_mail A-69  
 read\_mail 4-15  
 send\_mail 3-12

list\_original  
 active request  
 send\_mail A-71  
 request  
 send\_mail A-70

list\_requests request  
 read\_mail A-36  
 send\_mail A-71  
 read\_mail 4-13  
 send\_mail 3-10

log request  
 read\_mail 6-2, A-37  
 send\_mail 6-2, A-72

logbox 6-1  
 examining mail 6-2  
 storing mail 6-1

log\_original  
 request  
 send\_mail A-72

## M

mail  
 commands  
 print\_mail 2-1, A-2  
 read\_mail 4-1, A-6  
 recursion 7-6  
 send\_mail 3-1, 5-1, A-47  
 mail segment 6-5  
 suffix (.mail) 6-6

mailbox 1-2  
 active request  
 read\_mail 7-7, 7-8, A-37  
 commands 1-2, B-1  
 linking mailboxes 1-2  
 logbox 6-1  
 request  
 read\_mail A-37  
 saveboxes 6-3

mbrcr  
 see mbx\_create command

mbda  
 see mbx\_delete\_acl command

mbla  
 see mbx\_list\_acl command

mbsa  
 see mbx\_set\_acl command

mbx\_create (mbrcr) command B-3

mbx\_delete\_acl (mbda) command  
 B-4

mbx\_list\_acl (mbla) command  
 B-5

mbx\_set\_acl (mbsa) command  
 B-7

mbx\_specification  
 print\_mail A-2

message 1-3  
 commands  
 accept\_messages 1-2, 1-6  
 print\_messages 1-2

message (cont)  
   commands  
     send\_message 1-6  
     interactive 1-2, 1-6  
     numbers 4-2, 4-8  
     specifiers 4-4  
     ranges 4-5  
 message specifiers A-10  
 message\_id request  
   send\_mail A-72  
  
                                   N  
 next  
   active request  
     read\_mail A-37  
   keyword 4-4  
   request  
     read\_mail A-37  
  
                                   P  
 preface request 6-7  
   read\_mail A-37  
   send\_mail A-73  
 previous  
   active request  
     read\_mail A-38  
   keyword 4-4  
   request  
     read\_mail A-38  
 print request  
   read\_mail 4-2, 4-4, A-38  
   -idl 4-11  
   send\_mail 3-3, A-73  
 print\_header request  
   read\_mail 4-7  
   send\_mail 3-4, 5-2, A-73  
 print\_mail (prm) command 1-1,  
   2-1, A-2  
   control arguments  
   -list 2-3  
  
   print\_mail (prm) command  
     (cont)  
     control arguments  
     complete list A-2  
     responses 2-2, A-4  
 print\_messages (pm) command  
   1-2  
 print\_original request  
   send\_mail A-73  
 print\_original\_header request  
   send\_mail A-74  
  
   prm  
     see print\_mail command  
  
                                   Q  
 q request (editor) 3-5  
  
   qedx  
     editor 3-5  
     request  
       send\_mail 3-5, A-74  
 quit request  
   read\_mail 4-3, 4-11, A-39  
   send\_mail 3-9, A-75  
  
                                   R  
 ranges 4-5  
 rdm  
   see read\_mail command  
 re-entering mail system 7-6  
 ready request  
   read\_mail A-40  
   send\_mail A-76  
 ready\_off request  
   read\_mail A-40  
   send\_mail A-76



ready\_on request  
   read\_mail A-40  
   send\_mail A-76

read\_mail (rdm) command 1-1,  
   4-1, A-6  
   control arguments 4-15  
     -abbrev 7-1  
     -list 4-16  
     -no\_header 4-16  
     -request 7-4  
   complete list A-7  
   recursion 7-6  
   requests  
   see requests

recursion 7-6

Redistributed-By field 4-8

Redistributed-Date field 4-8

Redistributed-To field 4-8

remove request  
   send\_mail 5-5, A-76

reply request  
   read\_mail 4-7, A-40

reply\_to request  
   send\_mail A-77

request level  
   active requests 7-7

request loop 1-4  
   -request\_loop control  
     argument (sdm) 3-13  
   send\_mail 3-2, 4-7

requests 1-4  
   active requests 7-7  
   read\_mail  
     abbrev A-19, A-58  
     all A-20  
     current A-25  
     do A-27  
     execute 7-7, A-29  
     exec\_com A-28  
     first A-29  
     if A-33

requests (cont)  
   active requests  
     read\_mail  
       last A-33  
       list A-36  
       mailbox 7-8, A-37  
       next A-37  
       previous A-38  
       subsystem\_name A-45  
       subsystem\_version A-45  
     send\_mail  
       do A-64  
       execute 7-7, A-65  
       exec\_com A-65  
       if A-69  
       list\_original A-71  
       subject 7-7, A-78  
       subsystem\_name A-78  
       subsystem\_version A-78  
   print\_mail responses 2-2,  
     A-4  
   read\_mail  
     .request 7-6, A-18  
     .. escape 7-5, A-18  
     ? 4-12, A-18  
     abbrev A-19  
     all A-19  
     answer A-20  
     append 6-6, A-22  
     apply A-23  
     copy A-24  
     current A-25  
     delete 4-3, 4-8, A-25  
     do A-25  
     execute 7-8, A-28  
     exec\_com A-27  
     first A-29  
     forward 4-8, A-29  
     help 4-14, A-30  
     if A-32  
     last A-33  
     list 4-2, 4-3, 4-8, A-34  
       -idl 4-10  
     list\_help A-36  
     list\_requests A-36  
     log 6-2, A-37  
     mailbox A-37  
     next A-37  
     preface 6-7, A-37  
     previous A-38  
     print 4-2, 4-4, A-38  
       -idl 4-11

requests (cont)

- read\_mail
  - quit 4-3, 4-11, A-39
  - ready A-40
  - ready\_off A-40
  - ready\_on A-40
  - reply 4-7, A-40
  - retrieve 4-10, A-44
  - save A-45
  - subsystem\_name A-45
  - subsystem\_version A-45
  - write 6-6, A-45
- request\_line 1-5
- request\_loop 1-4, 3-2, 4-7
- send\_mail
  - . request 7-6, A-57
  - .. escape 7-5, A-58
  - ? 3-10, A-57
  - abbrev A-58
  - answer A-58
  - append 6-6
  - append\_path A-60
  - apply 7-9, A-60
  - cc 5-4, A-62
  - copy A-62
  - do A-62
  - execute 7-8, A-65
  - exec\_com A-64
  - fill A-66
  - from 5-7, A-66
  - help 3-11, A-66
  - if A-68
  - in\_reply\_to A-69
  - list\_help A-69
  - list\_original A-70
  - list\_requests A-71
  - log 6-2, A-72
  - log\_original A-72
  - message\_id A-72
  - preface 6-7
  - preface\_path A-73
  - print 3-3, A-73
  - print\_header 3-4, 5-2, A-73
  - print\_original A-73
  - print\_original\_header A-74
  - qedx 3-5, A-74
  - quit 3-9, A-75
  - ready A-76
  - ready\_off A-76
  - ready\_on A-76

requests (cont)

- send\_mail
  - remove 5-5, A-76
  - reply\_to A-77
  - save 6-3
  - save\_path A-77
  - save\_original A-77
  - send 3-7, 5-1, 5-3, 5-4, A-77
  - log and -save 6-4
  - subject 5-6, A-78
  - subsystem\_name A-78
  - subsystem\_version A-78
  - to 5-1, A-78
  - write 6-6, A-79
  - write\_original A-79
- retrieve\_request
  - read\_mail 4-10, A-44

S

- save\_request
  - read\_mail A-45
  - send\_mail 6-3, A-77
- savebox 6-3
  - examining 6-5
  - storing\_mail 6-3
- save\_original\_request
  - send\_mail A-77
- sdm
  - see send\_mail command
- segments 6-5
- selection\_control\_arguments
  - A-13
- send\_request
  - logging and saving 6-4
  - send\_mail 3-7, 5-1, 5-3, 5-4, A-77
- Sender field 5-7
- send\_mail (sdm) command 1-1, 3-1, 5-1, A-47

send\_mail (sdm) command (cont)  
control arguments 3-12  
-abbrev 7-1  
-acknowledge 3-14  
-comment 5-7  
-input\_file 3-13  
-log 6-2  
-request\_loop 3-13  
-save 6-3  
complete list A-48  
recursion 7-6  
requests  
see requests

T

text editor  
emacs 7-9  
others 7-9  
qedx 3-4  
  
to  
field 1-3  
request  
send\_mail 5-1, A-78  
send\_mail 5-2

send\_message (sm) command 1-6

W

start\_up.ec 1-6, 7-4

storing mail 6-1  
logbox 6-1  
saveboxes 6-3

write request  
read\_mail 6-6, A-45  
send\_mail 6-6, A-79

subject  
active request  
send\_mail 7-7, A-78  
field 1-3, 5-6  
of a message 3-1  
request  
send\_mail 5-6, A-78

write\_original  
request  
send\_mail A-79

subsystem\_name  
active request  
read\_mail A-45  
send\_mail A-78  
request  
read\_mail A-45  
send\_mail A-78

subsystem\_version  
active request  
read\_mail A-45  
send\_mail A-78  
request  
read\_mail A-45  
send\_mail A-78

suffix  
.mail 6-6  
.sv.mbx 6-1, 6-3

HONEYWELL INFORMATION SYSTEMS  
Technical Publications Remarks Form

CUT ALONG LINE

TITLE

MULTICS EXTENDED MAIL SYSTEM  
USER'S GUIDE

ORDER NO.

CH23-01

DATED

FEBRUARY 1983

ERRORS IN PUBLICATION

Empty box for reporting errors in the publication.

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Empty box for providing suggestions for improvement to the publication.



Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here.

FROM: NAME \_\_\_\_\_

DATE \_\_\_\_\_

TITLE \_\_\_\_\_

COMPANY \_\_\_\_\_

ADDRESS \_\_\_\_\_

\_\_\_\_\_

PLEASE FOLD AND TAPE—  
NOTE: U. S. Postal Service will not deliver stapled forms



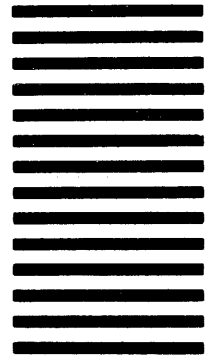
NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

**BUSINESS REPLY MAIL**  
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

**HONEYWELL INFORMATION SYSTEMS**  
200 SMITH STREET  
WALTHAM, MA 02154

**ATTN: PUBLICATIONS, MS486**



· CUT ALONG LINE ·

FOLD ALONG LINE

FOLD ALONG LINE

**Honeywell**

**Together, we can find the answers.**

**Honeywell**

**Honeywell Information Systems**

**U.S.A.:** 200 Smith St., MS 486, Waltham, MA 02154

**Canada:** 155 Gordon Baker Rd., Willowdale, ON M2H 3N7

**U.K.:** Great West Rd., Brentford, Middlesex TW8 9DH **Italy:** 32 Via Pirelli, 20124 Milano

**Mexico:** Avenida Nuevo Leon 250, Mexico 11, D.F. **Japan:** 2-2 Jinbou-Cho Kanda, Chiyoda-Ku Tokyo  
**Australia:** 124 Walker St., North Sydney, N.S.W. 2060 **S.E. Asia:** Mandarin Plaza, Tsimshatsui East, H.K.

36094, 1282, Printed in U.S.A.

CH23-01