# MH: A Multifarious User Agent

Marshall T. Rose
Member, Research Technical Staff
Northrop Research and Technology Center[†]

Einar A. Stefferud
President, Network Management Associates[‡]
and Visiting Lecturer, University of California, Irvine

Jerry N. Sweet
Member, Technical Staff
Local Network Systems[⋈]

Abstract

The UCI version of the Rand Message Handling System (MH) is discussed, including important extensions. MH is a powerful user agent which operates in the ARPA Internet and UUCP environments. In addition to the basic functions provided by a user agent, such as reading and sending mail, MH has several distinguishing characteristics which give the user additional message handling capabilities. In particular, MH provides mechanisms for organizing messages, tailoring its own behavior, and extending its functions.

This document describes MH from several perspectives. Particular emphasis is given to: the MH user environment, advanced features of MH which have proven to be particularly useful for sophisticated users of electronic mail, MH's potential as a record manager, and MH as a part of a distributed mail environment. Although MH as been widely used since its creation in 1979, a discussion of its perspectives and functionality has not appeared in the open literature.

---

[†] One Research Park, Palos Verdes Peninsula, CA 90274. Telephone: 213/377–4811.
Computer mail: `MRose%NRTC@USC-ECL`
[‡] 17301 Drey Lane, Huntington Beach, CA 92647. Telephone: 714/842–3711.
Computer mail: `EStefferud@ICS.UCI.EDU`
[⋈] 130 McCormick Avenue, Suite 102, Costa Mesa, CA 92626. Telephone: 714/754–6631.
Computer mail: `JSweet@ICS.UCI.EDU`

# MH: A Multifarious User Agent

## Introduction

The UCI version of the Rand Message Handling System, MH, is a user agent. In the interests of brevity, we dispense with the usual definition of terms, refer the reader to Figure 1, and simply note that MH is not responsible for delivering mail. Rather, it interacts with a *message transport system*, MTS, at two interfaces: it sends mail by placing it through a *posting slot* to the MTS, and it receives mail by retrieving it through a *delivery slot* from the MTS. Besides these two MTS-specific activities, the tasks which MH addresses are: the composition of messages (which may, or may not, be in reference to previously sent messages), the reading of messages, and the organization of messages.

MH was originally developed by the Rand Corporation, and initially was proprietary software. The Department of Information and Computer Science at University of California, Irvine, shortly after joining the Computer Science Network (CSnet), acquired a copy of MH, and began additional development of the software. Since that time, the Rand Corporation has declared MH to be in the public domain, and the UCI version of MH has passed through four major releases.

Much credit must be given to the initial designers and implementors of MH: Bruce Borden, Stockton Gaines, and Norman Shapiro. Although MH has suffered significant development at UCI since Rand's initial release, the fundamental concepts of MH's environs have remained nearly unchanged. In addition, the current maintainers of MH gratefully acknowledge the comments of the many sites which have run various releases of MH in the past.

MH runs on different versions of the UNIX[1] operating system (such as 4.2BSD UNIX and various flavors of v7 UNIX). In addition, MH supports four different MTS interfaces: SendMail[EAllm83], the standard mailer for 4.2BSD systems; MMDF[DCRoc79] and MMDF-II[DKing84], the Multi-Channel Memo Distribution Facility developed by the University of Delaware which forms the software-backbone for CSnet[DCome83] mail relays service; SMTP, the ARPA Internet Simple Mail Transfer Protocol[SMTP]; and, a stand-alone delivery system.

The organization of this paper is straight-forward, given space considerations. Initially, the MH philosophy of mail handling is presented, along with a description of the environment which the MH user is given to process mail. Following this, certain advanced features of MH are discussed in more detail. In particular, the

[1] UNIX is a trademark of AT&T Bell Laboratories.

Copyright © 1985, North Holland Publishing Company

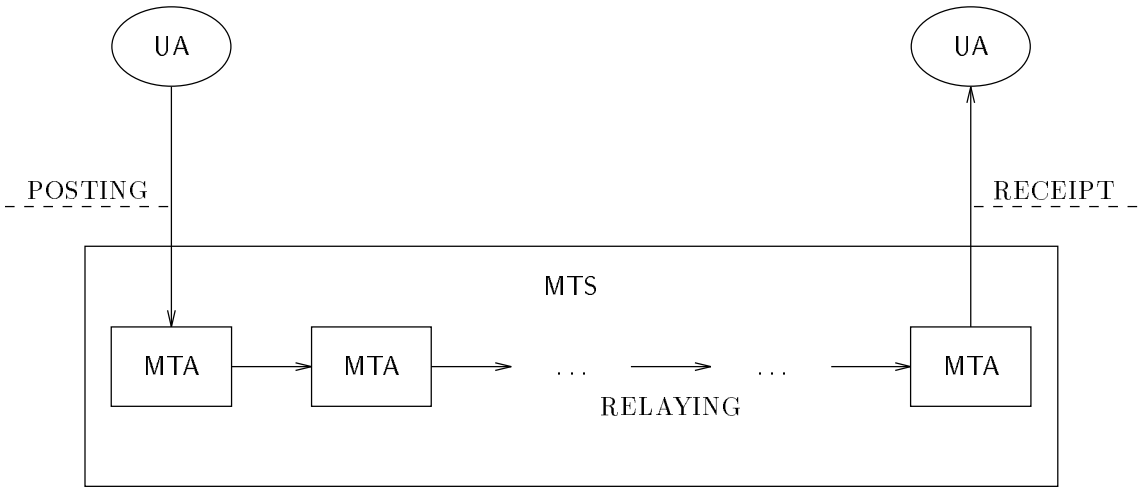


**Figure 1**
MTS model

notion of a *draft folder* is introduced, which permits the handling of multiple drafts during composition. In addition, message selection facilities are described. Next, two different aspects of MH's power as a software system are discussed: record handling, in which MH facilitates record processing systems; and, how MH can be employed in a distributed mail environment. This latter section raises questions as to the location of the posting and delivery slots, along with authentication mechanisms. Finally, we conclude by discussing areas of future development which MH may endure.

Although familiarity with MH is not assumed on the part of the reader, some knowledge of the UNIX operating system is useful. Appendix A gives a short synopsis of the MH commands.

## The MH Philosophy

Although MH has many traits which tend to differ it from other user agents, the design aspect which fundamentally influences the interface between MH and the user is that it is composed of many small programs instead of one very large one. This architecture gives MH much of its strength, since intermediate and advanced users are able to take advantage of this flexibility.

The key to this flexibility is that the UNIX shell (usually the *C* shell or the *Bourne* shell), is the user's interface to MH. This means that when handling mail, the entire power of the shell is at the user's disposal in addition to the facilities which MH provides. Hence, the user may intersperse mail handling commands with other commands in an arbitrary fashion, making use of command handling capabilities that the user's shell provides.

Furthermore, rather than storing messages in a complicated data structure within a monolithic file, in MH, each message is a UNIX file, and each folder (an object which holds groups of messages) is a UNIX directory. That is, the directory and file structure of UNIX is used directly. As a result, any UNIX file-handling command can be applied to any message.

To the novice, this may not make much sense or may not seem important. From three years of observation, we have seen that as users of MH have become more experienced they have found this capability to be quite attractive. In addition, this approach is often quite pleasing to system implementors, because it minimizes the amount of coding to be performed and, given a modular design, changes to the software system can be maintained easily. Our empirical findings confirm our theoretical expectations regarding the MH architecture.

Having described how MH fits into the UNIX environment, we now discuss the mail handling environment which is available to the MH user.

### *The* MH *Environs*

MH provides a complementary environment to the user's shell. While the shell maintains a context related to the user's focus in the file system (a *current working directory*), mail handling is performed in a separate mail folder context. Operations on mail can therefore be performed entirely without regard to the current file system context, although MH does not prevent the user from making use of that context. Certain mail handling functions do make use of information maintained by the shell. For instance, by setting certain shell parameters, called *environment variables*, alternate mail handling contexts can be selected.

MH conventions often have direct analogs to shell or file system conventions. The shell has a current working directory; MH has a current mail folder. When the user begins a session on the system, the user's "home directory" is the base context; MH's default base area, the Mail directory, is found under the user's home directory. The user's default shell parameters are set upon beginning a new session from a startup profile (called *.profile* for *sh* users or *.cshrc* for *csh* users); the default parameters for MH commands are taken from a file called *.mh_profile* in the user's home directory. The shell has an *environment*; MH has a *context* file. Each of the user's directories has files; each of the user's MH folders has messages.

These parallels have a basis not only in MH's high level mail handling model, but also in the way low level shell and file system conventions have been abstracted to implement MH conventions. Directories are folders; files are messages. The Mail directory forms the root of a virtual file subsystem within which the user operates on mail without disturbing files outside this mail handling domain.

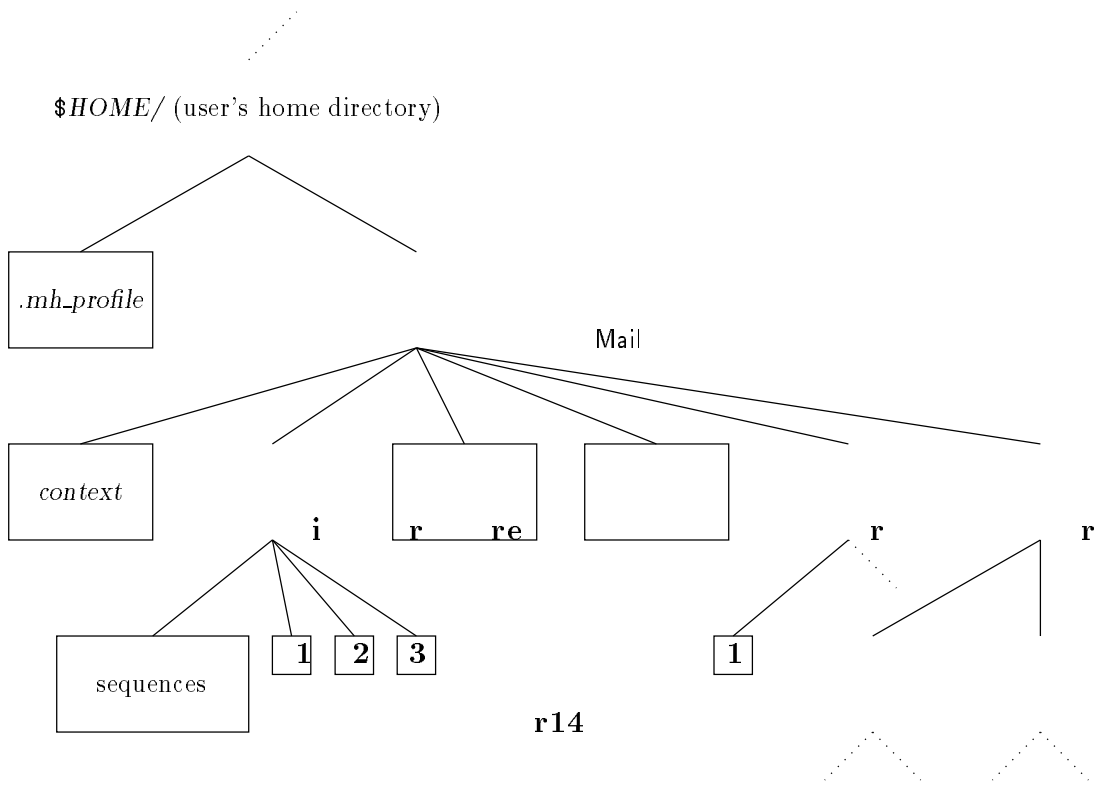


**Figure 2**
MH File Subsystem
(directories are shaded)

### *The* MH *Profile*

The *.mh_profile* contains plaintext that describes the user's default mail handling parameters. An example of an elaborated profile is shown in Figure 3.

Each line in the profile consists of an MH parameter name terminated with a colon (':') followed by parameter values. In this example, "global" parameters are listed in the first few lines, with program-specific parameters following. Each MH program examines global parameters as well as any parameter with the same name by which the program was invoked. For example, the *comp* program, which is used to compose new messages to be sent, examines the entries:

Path: The path parameter specifies the name of the MH root directory. This is normally named Mail.

Editor: The editor parameter specifies which text editor is first invoked to create the header information and body of a message draft. In most cases, this editor is the MH default editor, *prompter*.

Draft-Folder: This parameter specifies a folder within which new message drafts are to be created. The draft folder mechanism is an advanced

```
Path: Mail
Editor: prompter
prompter-next: emacs
Folder-Protect: 700
Msg-Protect: 600
Previous-Sequence: pseq
Alternate-Mailboxes: jsweet@uci-icse, jsweet@uci-750a
Draft-Folder: drafts
Sequence-Negation: not
bbc:      -quiet
bboards:  system mh-workers sf-lovers whimsey
comp:     -form mycomponents
dist:     -annotate -inplace
folder:   -noheader
forw:     -annotate -inplace -format
mhl:      -noclear
next:     -noheader
prev:     -noheader
prompter: -prepend
repl:     -annotate -inplace -cc me
send:     -format -msgid
scan:     -noheader -time
show:     -noheader -format
showproc: mhl
```

**Figure 3**
Elaborated MH Profile

feature of MH that is given separate treatment in a later segment of this paper.

comp: The program-specific parameter examined by *comp* lists user-default options.

Other programs invoked by *comp* (e.g. *prompter* and *send*) would examine their own profile entries as well. MH programs have reasonable compiled-in defaults and also permit options to be specified on the shell command line with which the programs are invoked. The order of override precedence is: command line options first, *.mh_profile* options second, and compiled-in defaults last.

Each program option is prefixed by a dash ('-') following the UNIX convention. Unlike most UNIX-style options, however, the options are words rather than single letters. An option may be abbreviated to an unambiguous prefix. Each MH program has a `'-help'` option that displays a brief summary of the program's available options.

### *Folders and Messages*

In a typical paper-oriented office, new correspondence arrives and is stacked in an "in box", while outgoing correspondence is placed in an "out box". Processed material is stored in appropriately labelled folders and filed away for future reference. This state of affairs is modelled in MH with *folders* and *messages*, which are simply text files (one message per file) stored under the folder directories. Most of the user's folders are kept under the Mail directory.

A folder is given an alphanumeric name permissible within the UNIX file system structure, and each message stored therein is given a numeric name in the range 1..1999. The upper bound on message numbers was selected for efficient access to an internal representation, an array of bits (a "bit set"), with each bit indicating the presence or absence of a message with a number in the range 1..1999. This internal representation also restricts the order of multiple message reference to an ascending numerical sequence. Other representations have been studied (e.g., an unsorted sparse array of integers), but have been rejected for reasons of efficiency. Folders may contain subfolders, corresponding to UNIX tree-structured directories. For the sake of completeness, it might be said that "sub-messages" exist insofar as message "digests", which nest messages inside other messages, are supported by certain advanced MH functions.

The current working folder is the default folder selected for almost all MH commands. To select explicitly a folder for mail handling commands entails specifying the name of the folder, prefixing the name with a plus-symbol ('+'). An example is:

```
refile␣1␣2␣3␣+chron/yr.1984
```

This command re-files the selected messages (*1*, *2*, and *3* here) from the current working folder to a subfolder under the folder *chron* named *yr.1984*. To see the folder/subfolder relationship, refer to Figure 2.

The plus-symbol notation is specific to those folders immediately subordinate to the Mail directory. This is analogous to "absolute pathnames" in UNIX—those files whose positions in the file system hierarchy are given starting with the system root, names prefixed with the slash character ('/'). To specify folders subordinate to the current working folder, an at-sign ('@') is substituted for ('+'). It is permitted to use UNIX dot notation to specify parent folders. Referring to Figure 2, if the current working folder were ``+chron/yr.1985'', then the command

```
folder␣@../yr.1984
```

selects the subfolder *yr.1984* in the parent directory *chron*, as the new current working folder. While the current working folder is normally the default, it may be specified explicitly as ``@.''.

```
To: <reply-to|from>
cc: <?to|cc|me><to>,<cc>,<me>
Subject: <?subject>Re: <subject>
In-reply-to: <?date><?message-id>Your message of <date>.
     <message-id>
In-reply-to: <?date><!message-id>Your message of <date>.
Fcc: <?fcc><fcc>
--------
```

**Figure 4**
Elaborated Reply Template

### *The Context File*

The *.mh_profile* contains static information about the user's preferences. A *context* file, contained in the Mail directory, contains the current mail handling environment information, which changes as different folders, messages, and named message lists (called *message sequences*) are selected, created, and updated. This information is retained between invocations of MH commands, and is preserved across system sessions.

### *Templates*

The message draft composition functions (*comp*, *repl*, *forw*, and *dist*) use certain default header formats, which may be changed by the user through the use of message templates. The exact format of a template may vary among commands. An example of an elaborated template for the reply command *repl* is shown in Figure 4.

This template specifies how the automatically-generated header for a draft message in reply to a source message is to be formatted. The syntax is capable of directing output of header lines based on the presence or absence of other header lines in the source message.

Other kinds of templates are used to specify the display formats of messages, or to specify the way that messages are to be included in other messages. This is similar to the functionality provided by BBN Hermes[HERMES], another powerful mail handling system for Tops20[2] based systems.

### *Explaining All This to New Users*

There do exist people who do not like MH.[3] The emerging pattern of complaints from such people indicates that MH accentuates their perceptions of the deficiencies of UNIX, to wit, lack of interactivity and lack of easily found help facilities. Also, some feel that the proximity of the mail handling environment to the operating system is a distraction, rather than an asset. There have been

[2] Tops20 is a trademark of Digital Equipment Corporation.

[3] At UCI, these people are reported to be weeded out at an early stage and quietly taken to the Ministry of Love to be made *uncrimethinkful*.

some attempts to make MH more accessible to users who prefer menu-oriented or monolithic mail system interfaces.[4]

In truth, users new to UNIX do not always acclimate to MH easily. The command set is undistinguishably mixed in with all other UNIX utilities, and it is not easy, without aid of a manual, to pick out the necessary commands. MH does not provide any "hand-holding" to guide the user through a minimally useful command subset.

Another problem is that the initial default user profile is too often sparse, containing only a ``Path:'' parameter. MH commands will perform adequately without specific information in the profile, so new users often neglect optionally useful MH capabilities, eventually becoming frustrated with the limited default capabilities, yet unable to determine without researching through the user's manual, the necessary options that would solve their problems.

The currently available means for learning how to use MH are:

- One-on-one tutoring by knowledgeable MH users, which has so far shown the best results with new users.
- Consulting the MH *Tutorial*[MRose84b], or the MH *User's Manual*[MRose85a].
- Using the *msh* ("MH shell") program as a training shell to read bulletin boards. The *msh* command is an interactive program that provides some help messages and can list available MH commands.

No on-line tutorial materials are presently distributed with the mh.5 system, although there are some plans in the works to provide a program to help with setting up the user profile that would also provide operational tips for MH and UNIX.

It should be noted that these perceived defects of MH do not affect its utility any more than analogous problems with any operating system will diminish its actual capabilities. Users may quarrel with the means chosen for orchestrating MH, but the fact remains that MH is a very useful set of mail handling tools that is flexible, infinitely interoperable with other UNIX text handling tools, and yet simple enough for new users to grasp once they are given the proper start. The fact that better tutorial materials and training do not exist only means that some further work needs to be done in the area of user-education.

[4] For example, *mhe* from Brian Reid of Stanford University and *emh* from Marshall Rose are instances of macro packages for James Gosling's EMACS extensible editor, while the *hm* program from Jim Guyton of the Rand Corporation is a monolithic MH interface. As of this writing, none of these programs is documented in the literature.

## A Few Advanced Features

We now consider certain advanced features in MH. These features have been chosen to demonstrate some useful capabilities available to the MH user. It should be noted that many capabilities of MH, such as shell scripts for extensibility, mail delivery hooks, the personal aliasing facility, and so forth, are not described here for lack of space.

### *Draft Folders*

The *draft folder* facility provides a method by which several message drafts can be simultaneously composed and maintained until sent. The rationale for this is that partially composed message drafts, perhaps elaborate sets of separate messages, can be incrementally completed, while a folder provides a consistent organization for drafts in progress. This is comparable to similar situations in the "paper world" where contracts, business correspondence, and other communications, rather than being created serially with each posted in turn before composing the next, are usually left in various stages of completion before they are eventually mailed.

The ``Draft-Folder:'' parameter value in the MH profile is used to specify a default draft folder, where each draft is given a number and an "artificial" date stamp. Provided that the proper header fields have been completed, a *scan* listing of the draft folder provides a summary of each draft in progress: to whom the message is to be sent, the subject, the date of the draft's initial creation and optionally, the current size of the draft in terms of characters. Experienced users of MH may often keep as many as five to ten unfinished drafts in their draft folder. "Draft clutter" can be remedied easily with the *rmm* command.

### *Message Selection*

MH commands accept *message sequence* specifications to specify which `msg` or `msgs` are to be operated upon. Here are some examples:

```
scan␣1␣3␣5␣19␣185
```

to get a scan listing of messages 1, 3, 5, 19 and 185.

```
scan␣pseq
```

to get a scan listing of whatever message sequence was given to the previous MH command (in this case 1, 3, 5, 19, and 185).

```
show␣first␣last
```

to get a display of the first and last messages in the folder. The MH sequences named ``first'' and ``last'' are system defined pseudo sequences which act like explicit sequences when given to MH commands. Others are ``cur'', ``next'', ``prev'', and ``all'' which respectively specify the "current" message, the "next" after cur, the "previous" message before cur, or "all" messages in the

current-folder. The *scan* assumes ``all'' while show assumes ``cur'', unless overridden on the command line. Over-ride precedence is: command-line first, *.mh_profile* second, and compiled-in default last.

Users can define additional sequences for similar use, but must avoid using reserved names. A few optional sequence names have been preempted by MH, such as ``pseq'' to mean the "sequence used by the previous MH command," and ``unseen'' to mean the "messages not yet seen by the user." Sometimes these preempted names can be changed by resetting them in the user's MH profile, but these facilities are beyond the scope of this discussion.

The mark command can be used to set the values for user-defined sequences:

```
mark␣1␣3␣5␣-seq␣zzz
mark␣4␣5␣9␣-seq␣zzz␣-nozero
```

will create a user-sequence named ``zzz'' and put the sequence ``1 3 5'' in it. The *mark* command assumes that any prior content in an existing user-sequence should be "zeroed" before the new sequence value is recorded. This can be prevented with a '-nozero' switch on the command line, to add ``4 5 9'' to the original ``1 3 5'' to yield ``1 3 4 5 9''.

```
mark␣pseq␣zzz␣-seq␣zzznew
```

will create a new sequence named ``zzznew'' and set its value to the combined (inclusive or) of the existing user-sequences in ``pseq'' and ``zzz'' for its value.

Another more powerful way to set the values of a user-sequence is with the pick command, which provides full string search capabilities:

```
pick␣-from␣mrose␣-seq␣yyy
pick␣-from␣mrose␣-seq␣yyy␣-list
```

will search though all the ``From:'' fields in the current folder for the string ``mrose'' and place the list of "hits" in the sequence named ``yyy''. The '-list' switch will cause the resulting list to also be displayed on the user's terminal. If no '-seq␣name' switch is given, pick will assume '-list' and will simply display the resulting list of hits on the user's terminal.

This '-list' behavior of pick allows users to take advantage of the UNIX backquoting facility to embed searches in other MH commands.

```
scan␣`pick␣-from␣mrose`
```

will produce a scan listing of '`-from␣mrose`' hits because the UNIX shell will spawn a process to execute the ''`pick␣-from␣mrose`'' segment and return the '`-list`' results as the message sequence to be scanned.

```
mark␣pseq␣-seq␣zzz
```

could then be used to capture the "previous sequence" in zzz for later use.

One last facility should be mentioned here. It is also possible to negate a sequence to specify a new sequence. The default negation string is ''`not`''.

```
scan␣notzzz
mark␣notzzz␣-seq␣zzznot
```

will give the user a scan listing of all the messages in the current folder that are not included in the sequence ''`zzz`''. The mark example will of course record the negation of zzz in zzznot. It is a bad idea to use the string ''`not`'' as the beginning of any user-sequence name, if ''`not`'' is defined as the negation string. (Users can choose a different negation string.)

From this discussion, it should be clear that MH provides a uniform set of ways to capture and use sequences to augment the user's short- and long-term memory and to manipulate lists of interesting messages. User-sequences are normally stored as RFC822 labeled text lines in a file (e.g., sequences) in the folder with the messages referred to in the sequence. If a user does not have write access to a folder, then the MH *mark* and *pick* commands will create a "private" sequence in the user's *context* file. Switches are available to give the user control over the choice of '`-private`' or '`-public`' sequence options.

Since user-sequences are stored as ordinary text lines in RFC822 labeled fields, there is no prohibition against someone writing programs to perform any kind of useful manipulation on MH sequences. Boolean operators can be implemented, or complex indexing structures could be developed to serve special purposes. If a DBMS can utilize UNIX pathnames or MH '`+folder`' and message names, then the full power of the DBMS might be applied. The intention of MH development teams has always been to leave open the widest possible array of options for later extension. The only restrictions should be the user's ingenuity, programming prowess, and the available machine resources. Unfortunately these resources always seem to be available in limited quantities.

### *Distribution Lists*

MH has a convenient interface to the UCI BBoards facility[MRose84a].[5] This facility permits the efficient distribution of interest group messages on a single

[5] The UCI BBoards facility can run under either the MMDF or SendMail, or in a more restricted form under stand-alone MH.

host, to a group of hosts under a single administration, and to the ARPA Internet community.

Described simply, an interest group is composed of a number of subscribers with a common interest. These subscribers post mail to a single address, known as a *distribution* address (e.g., `MH-Workers@UCI`). From this distribution address, a copy of the message is sent to each subscriber. Each group has a *moderator*, which is the person that runs the group. This moderator can usually be reached at a special address, known as a *request* address (e.g., `MH-Workers-Request@UCI`). Usually, the responsibilities of the moderator are quite simple, since the mail system handles distribution to subscribers automatically. In some interest groups, instead of each separate message being distributed directly to subscribers, a batch of (related) messages are put into a *digest* format by the moderator and then sent to the subscribers. Although this requires more work on the part of the moderator and introduces delays, such groups tend to be better organized.

Unfortunately, some problems arise with the scheme outlined above. First, if two users on the same host subscribe to the same interest group, two copies of the message will be delivered. This is wasteful of both processor and disk resources at that host.

Second, some groups carry a lot of traffic. Although subscription to a group does indicate interest on the part of a subscriber, it is usually not interesting to get 50 messages or so delivered to the user's private maildrop each day, interspersed with *personal* mail, that is likely to be of a much more important and timely nature.

Third, if a subscriber's address in a distribution list becomes "bad" somehow and causes failed mail to be returned, the originator of the message is normally notified. It is not uncommon for a large list to have several bogus addresses. This results in the originator being flooded with "error messages" from mailers across the Internet stating that a given address on the list was bad. Needless to say, the originator usually does not care if the bogus addresses got a copy of the message or not. The originator is merely interested in posting a message to the group at large. On the other hand, the moderator of the group does care if there are bogus addresses on the list, but ironically does not receive notification.

To solve all of these problems, the UCI BBoards facility introduces a new entity into the picture: all interest group mail is handled by a special component of the mail system. The distribution address maps to a special *channel* that performs several actions. First, if local delivery is to be performed, then a copy of the message is placed in a global maildrop for the interest group with a timestamp and a unique number. Local users can read messages posted for the interest group by reading this "public" maildrop. Second, if further distribution is to take place, a copy of the message is sent to the distribution address in such a way that if any of

the addresses are bogus, failure notices will be returned to the local maintainer of the group address list, rather than the originator of the message.

This scheme has several advantages: First, messages delivered to the local host are processed and saved once in a globally accessible area. The UCI BBoards facility supports software which allows a user to query an interest group for new messages and to read and process those messages in the MH-style. Second, once a host administrator subscribes to an interest group, each user can join or quit the list's readership without contacting anyone. Third, a hierarchical distribution scheme can be constructed to reduce the amount of delivery effort. Fourth, errors are prevented from propagating. When an address on the distribution list goes bad, the list moderator who is immediately responsible for the address is notified. If a local moderator does not exist, then the local PostMaster is notified (not the global group moderator).

In addition to solving the problems outlined above, the UCI BBoards facility supports several other capabilities. BBoards may be automatically archived in order to conserve disk space and reduce processing time when reading current items. Also, the archives can be separately maintained on tape for access by interested researchers.

Special alias files may be generated which allow the MH user to shorten address type-in. For example, instead of sending to `SF-Lovers@Rutgers`, a user of MH usually sends to ``SF-Lovers'' and the MH aliasing facility automatically makes the appropriate expansion in the headers of the outgoing message. Hence, the user need only know the name of an interest group and not its global network address.

Finally, the UCI BBoards facility supports *private* interest groups using the UNIX group access mechanism. This allows a group of people on the same or different machines to conduct a private discussion.

The practical upshot of all this is that the UCI BBoards facility automates the vast majority of BBoards handling from the point of view of both the PostMaster and the user.

MH provides three programs to deal with interest groups. The *bbc* program is used to check on the status of one or more groups, and to optionally start an MH shell on those groups which the user is interested in. The *bbl* program can be used to perform manual maintenance on a discussion group beyond the normal automatic capabilities of the UCI BBoards facility. Finally, the *msh* program implements an MH shell for reading BBoards, in which nearly all of the MH commands are implemented in a single program.

Observant readers may note that the use of *msh* is contrary to the MH philosophy of using relatively small, single-purposed programs. Sadly, the authors

admit that this is true. In an effort to avoid some problems with shared-access and message naming conventions (which are beyond the scope of this paper), BBoards are kept in maildrop format (monolithic) instead of folders. Some research has gone into overcoming this problem in order to restore MH's purity of purpose, but all solutions proposed to date are either unworkable or require significant recoding of MH's internals.

### *Encapsulation*

As described above, some interest groups appear in digest form. This means that the messages which appear in such a forum actually encapsulate other messages in their body. It turns out that the generation of a digest is not at all unlike the generation of a draft which forwards one or more messages. In RFC934[MRose85b], a method is proposed to standardize message encapsulation for the ARPA Internet community. MH uses this method for the generation of digests, forwardings, and blind-carbon-copies.

A key requisite for using an encapsulation technique for digests and forwardings is the ability to later decapsulate the contents. Without this ability, the forwarded messages are of little use to the recipients because they can not be distributed, forwarded, replied-to, searched-for, or otherwise processed as separate individual messages. In the case of a digest, a bursting capability is especially useful. Not only does the ability to burst a digest permit a recipient of the digest to reply to an individual digestified message, but it also allows the recipient to selectively process the other messages encapsulated in the digest.

For example, a single digest issue usually contains more than one topic. A subscriber may only be interested in a subset of the topic discussed in a particular issue. With a bursting capability, the subscriber can burst the digest, scan the headers, and process those messages which are of interest. The others can be ignored, if the user so desires.

Note that with proper encapsulation technology, one can argue for the re-distribution of messages simply becoming special cases of message forwarding. For example, the NBS Standard for Mail Interchange[FIPS98] and the recent CCITT draft on Mail Handling Systems standards[X.400] both discourage the re-distribution facility in favor of forwarding by encapsulation.

### *Encapsulation and Blind-Carbon-Copies*

Many user agents support a blind-carbon-copy facility. MH implements this using a form of encapsulation. It may not be apparent to the reader as to why encapsulation of the original message is a good way to deliver blind-carbon-copies. With a blind-carbon-copy facility, two types of addressees are possible in the draft to be sent: *visible* and *blind*. The visible recipients are listed as addresses in the ``To:'' and ``cc:'' fields, and the blind recipients are listed in the ``Bcc:'' fields of the draft. The idea behind this facility is that copies of the draft which are

delivered to the ``To:'' and ``cc:'' recipients should show the visible recipients only.

A major concern with a blind-carbon-copy facility is that blind recipients should be prevented from accidentally replying to the message in such a way that the visible recipients are included as addressees in the reply.

There are several methods to implement this facility. Most rely on posting two drafts with the MTS. One draft is destined for visible recipients, and simply lacks the ``Bcc:'' fields of the original draft. The second draft is destined for the blind recipients. The question then arises as to what form this latter draft posted should take.

One approach might be to disable the ``To:'' and ``cc:'' fields of the draft sent to the blind recipients (e.g., by prefixing the string ``BCC-'' to these fields). Unfortunately, this is often very confusing to the blind recipients because it differs from what the visible recipients got. Although accidental replies are not possible, it is often difficult to tell that the message received is the result of a blind-carbon-copy.

The method used by MH is to post two drafts, a visible draft for the visible recipients, and a blind draft for the blind recipients. The visible draft consists of the original draft without any ``Bcc:'' fields. The blind draft contains the visible message as a forwarded message. The headers for the blind draft contain the minimal RFC822 headers (``From:'' and ``Date:'') and, if the original draft had a "Subject:" field, then this header field is also included. In addition, MH alerts the recipient that the message is a blind-carbon-copy by placing this information in the initial encapsulation information in the blind recipient's copy. This scheme prevents inadvertent replies while allowing the recipient full access to an exact copy of what was sent to the visible recipients.

## MH as a Record Handler

Although message format standards such as RFC822 (and its predecessors) were originally devised to facilitate computer processing of interpersonal messages, there is no special reason why the concept should be limited to interpersonal message processing. Messages are just one of a variety of useful record forms that might be created in one place and transfered to another for processing. In this regard, RFC822 wisely left open the option for higher level applications to use arbitrary header names or field contents by proscribing MTS use of header names beginning with ``X-''.

MH carries though on this idea by allowing the *pick* command to accept any arbitrary field name for string searches, so MH users can select on any arbitrary field name without prior definition. Beyond this, since all messages are simply files

in UNIX directories, applications can be developed to apply any programmable process to any selected message.

For example, a *Time Card Form* might be called up by an MH user with

```
comp␣-form␣timecomps
```

to enter time and attendance information into ``X-time...:'' fields in a draft message record. The *timecomps* form would include the address of a supervisor who should validate the information, along with empty fields to be filled in with data. In fancy applications, this might be done with a sophisticated interactive data entry tool which would validate entered information, but this is an open choice within the MH framework. Another alternative would be to use a received message as the blank form to add a degree of central control over time and attendance reporting forms.

Receiving supervisors could simply register approval by using the MH *dist* command to resend subordinates' time cards to higher approval levels, or to send them to a time card collection address. The MH *dist* command automatically inserts "ReSent" header fields showing who resent it and the resending date. Alternatively, the MH *forw* command could be used to transfer a batch of approved time cards to the next processing station. If desired, a new "approval" command could be programmed to provide a more trusted authentication, perhaps with encryption of the content. Trusted mail systems, such as Trusted Mail[6] [MRose85c], are becoming available for this purpose.

At the final collection destination, an automated User Agent could be programmed to directly load the data into the Time and Attendance DBMS by parsing and decoding the data contained in the ``X-time...:'' fields. It might be noted that while the RFC822 does not restrict the internal forms of messages, it is necessary to conform to the interchange standard if specialized filters for message headers are not to be built to serve as *export laundries* (a term originating with Stephen H. Willson to describe conformance transformations in Ada[7]).

### *Mapping Between Record Modes (DBMS/MHS)*

This time and attendance example suggests that it is possible to define one-to-one mappings between RFC822 fields and DBMS data elements. For every DBMS data element definition, there is a potential corresponding RFC822 transferable equivalent definition which can facilitate mail transfers of record information. Indeed, a large portion of the definitional work is already done where a Data Base has already been defined. All that remains is to define the RFC822 equivalents.

---

[6] Trusted Mail is a trademark of Trusted Technologies, Incorporated.

[7] Ada is a trademark of the Department of Defense (Ada Joint Program Office).

The suggestion that a batch of time cards be forwarded inside a "cover" message implies that it is possible in the MH framework to recursively bundle messages within messages, and be able to recover the originals for separate processing at a receiving destination. The MH *burst* command can be applied recursively for this purpose because MH encapsulation uses an unambiguous scheme to delimit messages that are enclosed inside other messages. Thus, it should be possible to extract a structured set of records from a DBMS and mail the set to a foreign site for processing, or reinsertion into another DBMS. As long as the DBMS data element definitions correctly correspond to the RFC822 definitions, it is not even necessary for the source and destination DBMS systems to be the same.

From this discussion, it is concluded that the MH framework can be useful for building distributed record handling systems where people at widely scattered locations must create and submit record forms for processing at distant locations. This might prove to be especially effective when a mail system is also needed for other communication purposes. A network of sales offices is a good example, where general message service would be used for communications with remote manufacturing and distribution centers, and could also be used for an order entry system.

Another example might be for structured communications, as occur in requisition and purchasing systems. Requisitions could be filled in and mailed to approval offices, and resent or forwarded to others for action. At some point, the requisitions could flow into other other more suitable processing systems as needed. At the very least, the ability to originate requisitions can be distributed to anyone with access to a mail system that can originate a proper requisition form.

As a last example, MH already supports group discussions with its BBoard facilities which allow for automatic sorting of mail by group address, with shared private or public group access to contributed items. As has been shown to be possible with administrative record systems, there is no obvious limit to the ways that group discussion traffic might be organized into structured collections with indices, annotations, or reference pointers to aid in making conference archives more useful. Indeed, MH tools could even be used to feed discussion items into existing conference systems.

## Distributed Mail

Next, we consider how MH might be used in a distributed mail environment. Two schemes are discussed: one in which connectivity is high and connections are relatively "cheap", and one in which connectivity is low and connections are "expensive".

### *The ARPA Internet Environs*

The ARPA Internet community consists of many types of heterogeneous nodes. Some hosts are large mainframe computers, others are personal workstations. All communicate using the MILSTD TCP/IP protocol suite[IP, TCP]. Messages which conform to the Standard for the Format of ARPA Internet Text Messages[DCROC82] are exchanged using the Simple Mail Transfer Protocol[SMTP].

On smaller nodes in the ARPA Internet it is often impractical to maintain a message transport agent. For example, a workstation may not have sufficient resources (cycles, disk space) in order to permit an SMTP server and associated local mail delivery system to be kept resident and continuously running. Furthermore, the workstation could be off-net for extended periods of time. Similarly, it may be expensive (or impossible) to keep a personal computer interconnected to an IP-style network for long periods of time. In other words, the node is lacking the resource known as "connectivity".

Despite this, it is often desirable to be able to process mail with MH on these smaller nodes, and they often support a user agent to aid the tasks of mail handling. To solve this problem, a network node which can support a message transport entity (known as *service* host) offers a maildrop service to these less endowed nodes (known as *client* hosts). The Post Office Protocol[JREYN84] (POP) is intended to permit a workstation to dynamically access a maildrop on a service host to pick-up mail.[8] The level of access includes the ability to determine the number of messages in the maildrop and the size of each message, as well as to retrieve and delete individual messages. More sophisticated implementations of the POP server are able to distinguish between the header and body portion of each message, and send $n$ lines of a message to the POP client. This capability is useful in thinly connected environments where conservation of bandwidth is important. By utilizing a more intelligent POP client, a user may generate "scan listings" and dynamically decide which messages are worth taking delivery on. The philosophy of the POP is to put intelligence in the POP clients and not the POP servers.

The underlying paradigm in which the POP functions is that of a split-slot/remote-UA model. The client host (such as a workstation) is without a co-resident *message transport agent* (MTA), and thus makes use of a service host with an MTA to obtain posting (SMTP) and delivery (POP) services. The entity which supports this type of environment is called a remote-UA since the user agent resides on a different host than its associated message transport agent.

[8] Actually, there are three different descriptions of the POP. The first, cited in [JREYN84], was the original description of the protocol, which suffered from certain problems. Since then, two alternate descriptions have been developed. The official revision of the POP[MBUTL85], and the revision of the POP which MH uses (which is documented in an internal memorandum in the MH release). This paper considers the POP in the context of the MH release.

One very important issue which must be raised at this point is one of authentication. The POP requires that a client identify itself to the server using a server-specific user-id and a server/user-specific password. This authentication is required to prevent unauthorized entities from accessing a maildrop on a POP service host. It must be emphasized that the POP client is not a "trusted" entity of the MTS in any sense at all.

Ideally, one would also like to authenticate mail as it is posted on the POP service host using the SMTP. Currently, in the ARPA Internet community, no authentication is done with SMTP transactions. This is considered a shortcoming by those interested in researching the split-UA model of distributed mail. The MZnet environment, discussed in the next section, has authentication facilities for posting mail.

The current release of MH supports the above model fully: a POP client program is available to retrieve a maildrop on a POP service host. In addition, using the SMTP configuration for delivery in MH, a user is able to specify a search-list of service hosts (and networks) with which to try to post mail. Using this search-list, when an MH user posts a draft, the *post* program will attempt to establish an SMTP connection with each host in the list to post the message until it succeeds. Initial experimentation with the split-UA in a local network environment has proved quite successful.

### *The MZnet Environs*

In 1983, the MZnet project[ESTEF84] at the University of California, Irvine set out to study the problems involved with bringing Internet-class mail handling facilities to personal computers. The project used Apple II computers running the CP/M 2.2 operating system. Programming was done in a subset of the C language called BDS C. The transport system was based on the MMDF PhoneNet software, and implemented a *split-slot* arrangement between a personal computer and a larger, centralized mail distribution system that performed user authentication and provided a relatively secure mail transfer channel. The user agent, CP/MH, was based on MH.

A conclusion of the experiment was that small personal computer systems with dial-up phone connections constrain user agent systems design in ways that require use of a *split-slot* interface between the UA and its supporting MTA, and that this interface best provides the required services if it has error controlled command and data transfer facilities, with interactive behavior. Another conclusion indicated that a good design for a user agent in such a small personal computer environment could be based on a very modular architecture, such as MH. A final conclusion was that session-level authentication of the client UA is required for both posting and delivery.

It should be noted that the MZnet project had a profound influence on the development of the POP used by MH. A somewhat more detailed discussion of the relations between the two environments can be found in the POP description contained in the MH release.

## A Final Note

With the fifth major release of the MH system, it has become clear that most major increases in functionality can come only at the expense of either efficiency or portability. Although there has been great effort to keep MH portable to a number of UNIX implementations,[9] the divergence in process management facilities, file system enhancements, and even C compiler capabilities has already presented obstacles to some attempts to rehost the MH code.

There has been some discussion of implementing specialized MH daemons that maintain context information over one or more sessions, thus decreasing the amount of overhead involved in starting each MH command. Unfortunately, even if such daemons were to be implemented, they would be very difficult to move to versions of UNIX without sophisticated process management facilities, and even then the differences in "philosophies" of process management[WJoy83, EOlse84] would tend to keep such daemons system specific. A better solution seems to be simply to tune existing code.

## Acknowledgements

The authors would like to thank Norman Z. Shapiro and Phyllis Kantar of the Rand Corporation for their invaluable comments during the preparation of this paper.

## Distribution Information

For information concerning distribution mechanics for the current release of MH, please contact:

Support Group
Attn: MH Distribution
Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717 USA

714/856–6852

[9] As of this writing, there are approximately 75 sites running mh.5 on five different implementations of UNIX.

## References


[DCome83] D. Comer. The Computer Science Research Network CSnet: A History and Status Report. *Communications of the ACM 26*, 10 (October, 1983), 747–753.

[DCroc79] D.H. Crocker, E.S. Szurkowski, D.J. Farber. An Internetwork Memo Distribution Facility — MMDF. Appearing in *Proceedings, Sixth Data Communications Symposium*, Asilomar, 1979, pp. 18–25.

[DCroc82] D.H. Crocker. Standard for the Format of ARPA Internet Text Messages. Request for Comments 822. ARPA Internet Network Information Center (NIC), SRI International (August, 1982).

[DKing84] D.P. Kingston, III. MMDFII: A Technical Review. Appearing in *Proceedings Usenix Summer '84 Conference*, Salt Lake City, Utah, 1984, pp. 32–41.

[EAllm83] E. Allman. SENDMAIL — An Internetwork Mail Router. Britton-Lee, Inc., Berkeley, California (July, 1983).

[EOlse84] E.W. Olsen. NetOS Concepts and Facilities. Local Network Systems, Inc., Costa Mesa, California (August, 1984).

[EStef84] E.A. Stefferud, J.N. Sweet, T.P. Domae. MZnet: Mail Service for Personal Micro-Computer Systems. Appearing in *Proceedings, Second International Symposium on Computer Message Systems*, Nottingham, U.K, 1984, pp. 293–302.

[FIPS98] Specification for Message Format for Computer Based Message Systems. National Bureau of Standards (January, 1983).

[HERMES] Bolt, Beranek, and Newman. Hermes User's Manual. for TOPS-20. Bolt, Beranek, and Newman, Boston, MA (January, 1979).

[IP] Internet Protocol. Request for Comments 791 (milstd 1777). Appearing in *Internet Protocol Transition Workbook*, ARPA Internet Network Information Center (NIC), SRI International, 1981.

[JReyn84] J.K. Reynolds. Post Office Protocol. Request for Comments 918. ARPA Internet Network Information Center (NIC), SRI International (October, 1984).


Copyright © 1985, North Holland Publishing Company

[MButl85] M. Butler, J.B. Postel, et. al. Post Office Protocol - Version 2. Request for Comments 937. ARPA Internet Network Information Center (NIC), SRI International (February, 1985).

[MRose84a] M.T. Rose. The Rand MH Message Handling System: The UCI BBoards Facility. Department of Computer and Information Sciences, University of Delaware (October, 1984).

[MRose84b] M.T. Rose. The Rand MH Message Handling System: Tutorial. Department of Computer and Information Sciences, University of Delaware (October, 1984).

[MRose85a] M.T. Rose, J.L. Romine. The Rand MH Message Handling System: User's Manual. UCI Version. Department of Information and Computer Science, University of California, Irvine (January, 1985).

[MRose85b] M.T. Rose, E.A. Stefferud. Proposed Standard for Message Encapsulation. Request for Comments 934. ARPA Internet Network Information Center (NIC), SRI International (January, 1985).

[MRose85c] M.T. Rose, D.J. Farber, S.T. Walker. Design of the TTI Prototype Trusted Mail Agent. Appearing in *Proceedings, Second International Symposium on Computer Message Systems*, Washington, D.C., 1985 (to appear).

[SMTP] Simple Mail Transfer Protocol. Request for Comments 821. ARPA Internet Network Information Center (NIC), SRI International (August, 1982).

[TCP] Transmission Control Protocol. Request for Comments 793 (milstd 1778). Appearing in *Internet Protocol Transition Workbook*, ARPA Internet Network Information Center (NIC), SRI International, 1981.

[WJoy83] W.N. Joy, E. Cooper, R.S. Fabry, S.J. Leffler, K. McKusick, D. Mosher. 4.2bsd System Manual. Technical Report Number 5. Computer Systems Research Group, University of California, Berkeley.

[X.400] *Message Handling Systems: System Model-Service Elements*, Recommendation X.400, International Telegraph and Telephone Consultative Committee (CCITT).

# Appendix A
# MH Commands

MH is composed of several UNIX programs, which in theory are fairly simple and single-purposed. These commands are functionally grouped below:

Composing Mail

*comp*: compose a message
A program to originate a message. Usually, a special prompting editor front-end, *prompter*, is used to fill-in a composition template with the addressees of the message, subject, and so forth.

*dist*: redistribute a message to additional addresses
A program that re-enters a message previously received by the user into the message transport system. Only new addresses are added; the body of the message is not changed in any way.

*forw*: forward messages
A program that encapsulates one or more messages in a new message draft. In addition, the user may add initial and/or closing comments.

*repl*: reply to a message
A program that constructs a reply to a message using a reply template. The template mechanism has sufficient generality to permit the user to "program" the form of the reply draft based on the contents of the message being replied-to.

*send*: send a message
A program that posts a draft with the message transport system. The *send* program is usually invoked by one of the four preceding programs, and performs simple front-end pre-processing prior to invoking the *post* program. For example, if invoked in *push*'d mode, *send* will immediately relinquish control of the user's terminal and post the message in the background. If the posting fails, *send* will send back a failure notice to the user. If the user had *push*'d the sending of the draft, then by default the draft being sent is encapsulated in the failure notice. This permits easy *burst*'ing of the failure notice to retrieve the original draft. Otherwise, if the posting was successful, the draft is marked as having been sent.

*whatnow*: prompting front-end for send
A program which is called by *comp*, et. al., after the initial draft has been

Copyright © 1985, North Holland Publishing Company

generated. The MH user can specify a different *whatnow* program, which yields considerable extensibility.

*whom*: report to whom a message would go
A program which examines the addresses of the draft and expands all user-defined aliases contained therein. Optionally, *whom* may actually interact with the message transport system to determine the validity of the final addresses. This program is also usually invoked by *comp*, et. al.

Posting Mail

*ali*: list mail aliases
A simple front-end to the MH aliasing mechanism.

*ap*: parse addresses 822–style
A useful debugging tool for PostMasters who wish to examine how MH interprets an Internet address.

*conflict*: search for alias/password conflicts
Another program used by system administrators to check the consistency of MH alias files, and portions of the local message transport agent.

*install-mh*: initialize the MH environment
A program which is automatically executed the first time a user issues an MH command. This program performs once-only initialization of the user's MH environment.

*mhmail*: send or read mail
A simple program generally used by other programs to generate messages. The *mhmail* command is similar in purpose to the old *BellMail* program.

*post*: deliver a message
A complex MH back-end that interacts with the local message transport agent to enter messages through the posting slot. (See the description of *send* above).

Reading Mail

*inc*: incorporate new mail
A program that interacts with the local message transport agent to retrieve messages from the user's maildrop.

*msgchk*: check for waiting mail
A program which reports the status of mail waiting in the user's maildrop.

*show*: show (list) messages
A program which lists messages to its standard output (usually the user's terminal), possibly invoking another program to do the actual listing. Most users of MH have *show* automatically call the *mhl* program to format the

message. The *next* and *prev* programs are simply ``show␣next'' and ``show␣prev'', respectively.

*mhl*: produce formatted listings of MH messages
A program which displays a message as directed by a template. This permits the user to filter out uninteresting headers and re-arrange other headers to a particular preference. In addition to being invoked by *show*, the *mhl* program is optionally also invoked by *forw* to format each message being forwarded; invoked by *repl* to format the body of a message being replied-to, if that message is being included in the reply draft; and, invoked by *post* to format a message being sent as a blind-carbon-copy.

*rmm*: remove messages
A program that removes messages from an MH folder, optionally running a user-defined program instead of deleting them. If no program is given, the messages are "softly" removed, so they may possibly be recovered later.

*scan*: produce a one-line-per-message scan listing
A program that generates a scan listing for messages. Each line of the listing contains date, source, subject, and possibly the initial body of the message.

Folder Handling

*folder*: set/list current folder/message
A program used to list information concerning the current folder, or set the current folder and/or message.

*folders*: list all folders
A program to list information on all folders (actually, just a special case of the *folder* command). Since the MH folder structure may be recursive, the user can indicate that *folders* should recursively examine all folders.

*refile*: file message(s) in (an)other folder(s)
A program to move (or copy) messages from a source folder to one or more destination folders.

*rmf*: remove folder
A program that deletes a folder and all messages therein.

Message Selection

*anno*: annotate messages
A program to arbitrarily annotate messages. If the user so desires, after distributing, forwarding, or replying-to a message, MH will automatically attach an annotation to the original message indicating the date and addresses.

*mark*: mark messages
A program to manipulate user-defined sequences (lists of messages). Usually, *mark* is not employed directly by the MH user.

*pick*: select messages by content
A program to examine a list of messages and choose those which meet a particular selection criterion. The *pick* program is often used in UNIX back-quoted operations to pass message sequences to other MH commands.

*sortm*: sort messages
A program to sort a list of messages according to the date given in a particular field.

Distribution List Handling

*bbc*: check on BBoards
A front-end to run *msh* on a list of distribution lists which the user isn't current on.

*bbl*: manage a BBoard
A (depreciated) program used to manually manage the local archives of a distribution list. These functions (archiving, expunging) are performed automatically by MH.

*burst*: explode digests into messages
A program used to decapsulate messages from ARPA Internet digests. In addition, messages which have been encapsulated during forwarding (i.e., with *forw*) can also be decapsulated using *burst*.[10]

*msh*: MH shell (and BBoard reader)
A monolithic program used to implement MH commands on messages arranged in a single file (maildrop format). Useful since distribution lists are kept in this format to minimize consumption of system resources.

*pack*: compress a folder into a single file
A program which takes messages stored in MH format and places them in a single file (using the same format known by *msh*).

Interface to the UNIX File System

*mhpath*: print full pathnames of MH messages and folders
A program which maps MH-style names into the UNIX file naming convention.

[10] Similarly, blind-carbon-copies may be decapsulated, though only socially mature users should do so.

# Contents


| | Page |
|---|---|
| Introduction | 1 |
| The MH Philosophy | 2 |
| The MH Environs | 3 |
| The MH Profile | 4 |
| Folders and Messages | 6 |
| The Context File | 7 |
| Templates | 7 |
| Explaining All This to New Users | 7 |
| A Few Advanced Features | 9 |
| Draft Folders | 9 |
| Message Selection | 9 |
| Distribution Lists | 11 |
| Encapsulation | 14 |
| Encapsulation and Blind-Carbon-Copies | 14 |
| MH as a Record Handler | 15 |
| Mapping Between Record Modes (DBMS/MHS) | 16 |
| Distributed Mail | 17 |
| The ARPA Internet Environs | 18 |
| The MZnet Environs | 19 |
| A Final Note | 20 |
| Acknowledgements | 20 |
| Distribution Information | 20 |
| References | 21 |
| Appendix A: MH Commands | 23 |


This document (version #1.54) was TEXset April 12, 1990 with DISS.STY v103.