MH.5: How to process 200 messages a day and still get some real work done[⋈]

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

John L. Romine
Computing Support Group
Department of Information and Computer Science
University of California, Irvine[‡]

Abstract

The UCI version of the Rand Message Handling System (MH) is discussed. MH is a powerful user agent which operates in the ARPA Internet and UUCP environments. In addition to the usual functions provided by similar programs, MH has several distinguishing characteristics which give the user additional message handling capability. In particular, MH provides mechanisms for maintaining an organized mail environment, tailoring its 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, the user interface issues in MH, and the mh.5 distribution. The paper concludes with a summary of the authors' experiences with MH, and a discussion of future areas of enhancement.

[⋈] Alternate title: *MH: Your Key to Success.*

[†] One Research Park, Palos Verdes Peninsula, CA 90274. Telephone: 213/377–4811.
Computer mail: MRose%NRTC@USC-ECL, ...!{ucbvax!ucivax,trwrb}!nrtc!mrose.

[‡] University of California at Irvine, Irvine, CA 92717. Telephone: 714/856–6852.
Computer mail: J-Romine@USC-ECL, ...!{ucbvax,trwrb}!ucivax!jromine.

MH.5:
How to process 200 messages a day and still get some real work done

Introduction

The UCI version of the Rand Message Handling System, MH, is a software system that performs two functions: first, it interfaces a user to a message transport system, so the user may receive and send mail; second, it permits the user to maintain an organized mail environment to facilitate the composition of new messages and the reading of old messages. In short, while not responsible for the delivery of messages, MH aids the user in handling mail.

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. The current version, mh.5, is available from U.C. Irvine for a nominal distribution fee, or may be retrieved from the University of Delaware via anonymous FTP.

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 authors of the current release gratefully acknowledge the comments of the many sites which have run various releases of MH in the past. In particular, the dozen or so beta test sites for mh.5 provided tremendous help in stabilizing the current release.

MH runs on different versions of the UNIX[1] operating system (such as Berkeley 4.2BSD and various flavors of v7). In addition, MH supports four different message transport 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 relay service; SMTP, the ARPA Internet Simple Mail Transfer Protocol[SMTP]; and, a stand-alone delivery system.

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

This paper is organized in a straight-forward fashion: 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, such as facilities for selecting messages, and "advanced" concepts in *draft* handling. In addition, user interface issues in mail handling are addressed, and the merits of MH's approach is critically examined. Next, the mh.5 distribution package is described. Finally, we conclude by discussing the authors' experience with MH development and introducing areas where MH may be further developed.

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 distinguish it from other systems which handle mail, there is a single fundamental design decision which influences the interface between MH and the user: MH differs from most other systems in 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 which the user's shell provides.

Furthermore, rather than storing messages in a complicated data structure within a monolithic file, each message in MH is a UNIX file, and each folder (an object which holds groups of messages) in MH 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. However, as users of MH become more experienced, they find this capability 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. There are, however, performance penalties to be paid with this scheme. This issue is considered later in the paper.

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*

In the *$HOME* directory of each MH user, a file named *.mh_profile* contains static information about the user's MH environment, and default arguments for MH programs. For the latter case, each line of profile takes the form:

```
program-name:␣options
```

Each MH program consults the user's *.mh_profile* for its options. These options are consulted prior to evaluating any command-line arguments, and so provide the MH user the capability to customize the defaults for each command. Futher, by using the UNIX link facility, different names can be given to the same command. Since each MH command looks in the *.mh_profile* for a component with the name by which it was invoked, it's possible to have different defaults for the same program. For example, it is not uncommon to link *prompter* (a simple prompting editor front-end) under the name *rapid* in the user's *bin/* directory, and add to the *.mh_profile*:

```
rapid:␣-prepend␣-rapid
```

As a result, when *prompter* is invoked as *rapid*, it automatically uses the '`-prepend`' and '`-rapid`' options.

The profile component ''`Path:`'' is the path to the user's MH-directory, usually Mail. In addition to containing the user's folders, the MH-directory also contains *skeletons* and *templates* used by the MH programs, and the user's *context* file. This latter file has the same format as the user's *.mh_profile*, and contains the dynamic, context-dependent information about the user's environment. Whenever MH looks for an MH-specific file, such as a template or skeleton, it first consults the user's MH-directory, and then a system-wide library area.

The MH user always has a *current folder*, which is the folder in which the user is currently (or was last) working. Since any MH program which deals with folders implicitly manipulates this information, the name of the current folder is stored in the *context* component ''`Current-Folder:`''. Every folder has a *current message* known as '`cur`'. These values are the defaults for MH commands which accept folder and/or messages arguments.

MH programs make use of a set of envariables which further customize their behavior. The *$MH* envariable, if present, specifies the name of an alternate profile for the user. This allows a user of MH to easily maintain multiple mail-handling environments.

In terms of command syntax, most MH commands accept an optional *folder* argument, such as '`+outbox`'. Unlike most UNIX commands, all MH commands have switches which are words, rather than single letters. Switches may be abbreviated to the least unambiguous prefix. All MH commands also support

```
% inc
Incorporating new mail into inbox...

   1+ 03/16 Rand MH System    MH transcript  <<Here's the body of a sample m

% show
(Message inbox:1)
To: jromine@uci-icsa
Subject: MH transcript
Date: 16 Mar 85 18:28:59 PST (Sat)
From: Rand MH System <mh@uci-icsa>

Here's the body of a sample message.
% repl
To: Rand MH System <mh@uci-icsa>
cc: jromine@uci-icsa
Subject: Re: MH transcript
In-reply-to: Your message of 16 Mar 85 18:28:59 PST (Sat).
--------
Thanks for the test.

/JLR
^D

What now? send
% comp
To: MRose@UCI
cc:
Subject: sample comp
--------
Here's a sample compose for the MH transcript.

/JLR
^D

What now? send -verbose
 -- Posting for All Recipients --
  -- Local Recipients --
  MRose: address ok
 -- Recipient Copies Posted --
Message Processed
```

Figure 1
An MH Session

a ‘`-help`’ switch, which lists the syntax of the command along with available switches, and the version number of the command. Most MH commands also take a ‘`msg`’ or ‘`msgs`’ argument which takes the form of a message number (‘‘`1`’’), a message range (‘‘`1-2`’’), a standard sequence name (‘‘`cur`’’), or a user-defined sequence name (‘‘`select`’’).

An MH *Transcript*

Figure 1 contains a transcript of a simple MH session. First, *inc* is run to incorporate the new mail into the user's `''+inbox''` folder.

A *scan* listing of the mail is printed while it is being incorporated. (The user could run *scan* explicitly to generate additional *scan* listings later on.) The *scan* listing gives the message number, followed by the date, message sender, and subject. (If the message originated from the user generating the listing, the `''to:''` addressee is displayed instead of the sender.) If the subject is short, the first part of the message body is displayed after the characters `''<<''`. The plus sign ('+') after the message number indicates the current message.

The user *show*s the message, and decides to *reply*. A reply draft is created using the headers of the message being replied-to, using the default *replcomps* template. The default editor, *prompter*, is called to edit the draft. When an EOT is typed, *prompter* exits and the user is left at the `What now?` prompt. The option *send* is chosen. Since there were no problems in posting the draft with the message transport system, no additional output is produced. (MH is not verbose by default.)

The user then decides to compose a new message. The default skeleton, *components*, is copied to the draft, and *prompter* is once again called. After entering the addresses, subject, and body, the user then *send*s the *draft* from the `What now?` prompt, using `''send␣-verbose''`, which causes MH to list out the message addresses as it submits them to the message transport system.

Some MH Features

We now consider certain advanced features in MH. These features have been chosen to demonstrate some useful capabilities available to the MH user.

Message Sequences and Selection

MH has several built-in message sequence names, which may be used anywhere a `'msg'` or `'msgs'` argument is expected. These are: `'cur'`, `'next'`, `'prev'`, `'first'`, `'last'`, and `'all'`. Message ranges may also be specified. For example, `'all'` is actually `'first-last'`, and `'+mh␣last:5'` references the last five messages in your `'+mh'` folder. A powerful capability of MH is the ability to use not only the pre-defined message sequence names, but also arbitrary user-defined message sequence names.

Although all MH programs recognize user-defined sequences when appropriate, the *pick* and *mark* commands can create and modify user-defined message sequences. The *mark* command allows low-level manipulation of sequences, and is not particularly interesting in our discussion.

The *pick* command selects certain messages out of a folder. The criteria used for selection may be a search string and/or a date range.

Searching is performed on either a specific header in the message (e.g., `‘‘To:’’`), or anywhere within the message. By default, *pick* lists out the message numbers that matched the selection criteria. Thus, *pick* is useful in backquoted operations to the shell. For example, to scan all the messages in the current folder from “frated”, the MH user issues the command:

```
scan `pick -from frated`
```

To perform more complicated message selection, user-defined sequences are employed. Supplying a `‘-sequence name’` argument to *pick*, will cause it to define the sequence `‘name’` as those messages matched.

Giving *pick* a list of messages causes it to limit its search to just those messages. For example, to find all the messages in the current folder from “frated” also dated before friday:

```
pick -from frated -sequence select
pick select -before friday -sequence select
```

With the first *pick* command, the sequence `‘‘select’’` is defined to be all those messages from “frated”. In the second command, only those messages already in the `‘‘select’’` sequence are searched, and the `‘‘select’’` sequence is redefined to be only those messages which are also dated before friday. Those messages could then be *shown* with:

```
show select
```

When a `‘-sequence name’` argument is given to *pick*, the default behavior — listing the message numbers matched — is inhibited. To re-enable this behavior, the `‘-list’` option may be given. As a result, advanced users of MH often put the following line in their *.mh_profile*:

```
pick: -sequence select -list
```

which allows them to easily make use of the `‘select’` sequence as the messages last selected with *pick*.

Often it is desirable to act upon those messages which are *not* members of a given sequence. For this purpose, the `‘‘Sequence-Negation:’’` profile entry is useful. If the name of a user-defined sequence is prefixed with the value of the sequence-negation profile entry, MH commands will operate upon those messages which are *not* members of that sequence. For example, given a profile entry of:

```
Sequence-Negation: not
```

those messages which are not in the `‘select’` sequence could be *scan*’d with:

```
scan notselect
```

Obviously, some confusion could result if an attempt was made to define a sequence name which began with the sequence-negation string (e.g., `''notselect''`). For this reason, MH users will often use a single character, which their shell doesn't interpret, as their sequence-negation string (e.g., up-caret ('^') for *C* Shell users, and exclamation-mark ('!') for *Bourne* shell users).

MH also provides a way of automatically remembering the last message list given to an MH command. This facility is implemented by using a profile entry called `''Previous-Sequence:''`.

Draft Handling

After the initial edit of a message draft, the *comp*, *dist*, *forw*, and *repl* programs give the user a `What now?` prompt. The valid responses include: *edit* to re-edit the draft, *quit* to exit without sending the draft, *send* to send the draft, and *push* to send the draft in the background.

When the *send* option is given, the draft is posted with the message transport system. If there problems posting the draft, the `What now?` prompt is re-issued, so errors in the draft may be corrected.

Since posting the draft can be slow, the *push* option allows the MH user to send the draft in the background, and return immediately to the shell. If there are problems posting the message, the user will not see the diagnostics produced by the message transport system. For this reason, if *push* is used instead of *send*, and the message is not successfully posted, MH mails a message to the user containing any diagnostics which the message transport system produced along with a copy of the message. Later, the draft may be re-edited by entering `''comp␣-use''`.

A relatively new feature of MH is the ability to use a folder to store multiple drafts. These drafts are kept in an ordinary MH folder, and may be operated upon by MH commands. To enable this feature, the MH user selects a folder-name for the draft-folder, and creates an entry in the *.mh_profile*:

```
Draft-Folder:␣+foldername
```

From this point on, when a message is composed, the draft will be created as a message in that folder, instead of using the *draft* file in the user's MH directory. Unfortunately, if posting problems occur on a message which has been *push*'d, it may be difficult to re-edit the draft with `''comp␣-use''`. This might be the case if the user had started composing another message, while that first draft was being posted. In that event, the current-message in the draft-folder would no longer point to the failed draft.

There is a solution for this problem, however. By default, *push* assumes the `'-forward'` option, which says that if the message draft fails to be posted, it should be forwarded back to the user in the error report which *push* generates. The failed draft may then be extracted with the *burst* program (discussed later).

BBoards

MH has a convenient interface to the UCI BBoards facility[MRose84a].[2] This facility permits the efficient distribution of interest group messages on a single host, to a group of hosts under a single administration, and to the ARPA Internet community.

Although most readers are probably familiar with the concept of an interest group in the Internet context, a brief description is now given. Observant readers will notice that the distributed nature of the "network news" (a.k.a. USENET) tends to avoid many of the problems described below.

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 the *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*, who is the person that runs the group. This moderator can usually be reached at a special address, known as the *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 (hopefully related) messages are put into a *digest* format by the moderator and then sent to the subscribers. (This is similar to a newsletter format.) 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 are 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 or so messages delivered each day to the user's private maildrop, interspersed with *personal* mail, which 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

---

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

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 these problems, the UCI BBoards facility introduces a new entity into the picture: a *distribution channel.* All interest group mail is handled by the special mail system component. The distribution address for an interest-group maps mail for that interest-group to the distribution channel, which then performs several actions. First, if local delivery is to be performed, 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 may 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. Finally, errors are prevented from propagating. When an address on the distribution list goes bad, the list moderator who is 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 entry. 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 manually perform 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-purpose programs. Sadly, the authors admit that this is true. In an effort to minimize use of system resources however, BBoards are kept in maildrop format instead of folders.[3] Some research has gone into overcoming this problem to restore MH's purity of purpose, but all solutions proposed to date are either unworkable or require significant recoding of MH's internals.

Bursting

Internet interest group mail is often sent out in digest form. The experienced MH user may wish to deal with the digest messages on an individual basis, however. The *burst* program allows the MH user to extract these digest messages, and store each as an individual MH message.

Burst will also extract forwarded messages generated by *forw* (or the forwarded message in the error report generated by *push*, as described above). Although *burst* cannot always decapsulate messages encapsulated by sites not running MH, it adheres to the proposed standard described in [MRose85b].

[3] When the message transport system delivers a message to a user it stores it in a single file, called a *maildrop*. Since many messages may be present in a single maildrop, (in theory) there is a unique string acting as a separator between messages in the maildrop. Although this is convenient for storage of messages, it makes retrieval more difficult unless a separate index into the maildrop is kept. This latter approach is taken by the *msg* program available with MMDF-II and by *msh* as well.

Distributed Mail

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 system (e.g., SendMail). 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 manage 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.[4] 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 decide dynamically 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 current release of MH supports the above model fully. A POP client program is available to retrieve a maildrop from a POP service host. In addition, using the SMTP configuration for delivery in MH (either in conjunction with SendMail or the MMDF), a user is able to specify a search-list of service hosts (and/or networks) 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 search-list to post the message until it succeeds. Initial

[4] 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.

experimentation using the POP and MH in a local network environment has proved quite successful.

User Interface Issues in MH

At this point, it is perhaps useful to take a step backwards and examine the success and problems of MH's approach to user interfaces.

Creeping Featurism

A complaint often heard about systems which undergo substantial development by many people over a number of years, is that more and more options are introduced which add little to the functionality but greatly increase the amount of information a user needs to know in order to get useful work done. This is usually referred to as *creeping featurism.*

Unfortunately MH, having undergone six years of off-and-on development by ten or so well-meaning programmers (the present authors included), suffers mightily from this. For example, the *send* command has twenty-five visible switches, and at least nine hidden switches, for a total of thirty-four. The poor user who types

```
send␣-help
```

watches the options scroll off the screen (since the '`-help`' switch also lists out four other lines of information).[5] The sad part is that all of these switches are useful in one form or another.

There are a lot of good things to be said for the "one program, one function" philosophy of system design. In the MH case, however, each program really does only one mail handling activity (with a few minor exceptions). The options associated with each command are present to modify the program's behavior to perform similar, but slightly different tasks. In further defense of MH, note that there are 32 MH commands at present, all performing different tasks.

The problem with creeping featurism though, is that while the functionality of the system increases sub-linearly, the complexity of the system increases linearly. That is, although the number of switches that a program takes might double, it is unlikely that the program's functionality or capabilities will double.

[5] Recently, this was fixed by compressing the way in which switches are presented. The solution is only temporary however, as *send* will no doubt acquire an *endless* number of switches in the years to come.

```
To:
cc:
Bcc:
Fcc: outbox
Fcc:
Subject:
Reply-To:
--------
```

Figure 2
Draft Skeleton

```
To: <reply-to|from>
cc: <?to|cc><to>,<cc>
Fcc: +outbox
Fcc: <?fcc><fcc>
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>.
--------
```

Figure 3
Reply Template

Templates versus Switches

One way to trim the explosion of available options, while still increasing functionality, is to introduce options with a richer domain. Hence, instead of using options which take *on* or *off* forms or simple numeric or string values, the possible values which an option might take on is given a large space. There are several ways that this might be accomplished.

The *comp*, *dist*, and *forw* programs use draft *skeletons* (simple form fill-in files) to construct the general format of the draft being composed. An example of a draft skeleton used for composing new messages (by *comp*) is shown in Figure 2. The approach is to let the user specify (and later edit) both arbitrary headers of draft and the body of the draft. Note while most of the fields are empty, the first ``Fcc:'' field already contains a value. By using the simple prompting editor, *prompter*, the user can speedily enter the headers of the message. The *prompter* program given the skeleton in Figure 2 would prompt the user for the contents of each field, except for the second ``fcc:'', which it would include verbatim. It would then read the body of the message up to an end-of-file. Naturally, the MH user is free to use *any* editor to edit *any* part of the draft (headers or body). This example demonstrates the flexibility achieved by not limiting what headers a draft may contain (which most mail sending programs do), while still retaining the simplicity of being able to treat the entire message draft as a UNIX file.

```
From: <?me>Message Agent \<<me>>
To: <reply-to|from>
Fcc: +rcvtrip
Fcc: <?fcc><fcc>
Subject: <?subject>BEEP! Re: <subject>
Subject: <!subject>BEEP!
In-reply-to: <?date><?message-id>Your message of <date>.
     <message-id>
In-reply-to: <?date><!message-id>Your message of <date>.
--------

    This is an automatic reply.  Feel free to send additional mail, as only
    this one notice will be generated.

    I am attending the USENIX Summer '85 conference in Portland, Oregon.
    I expect to be reading mail again on the 16th of June.

/mtr
```

Figure 4
The *tripcomps* Reply Template

Another more interesting approach is used by the *repl* command, which constructs a draft in reply-to a previously received message. Instead of adding switches to indicate which fields of the draft should be derived from the message being replied-to, and how they should be derived, a single option, the ability to specify a *template*, was made available. An example of a reply template is shown in Figure 3. Put simply, based on the presence of certain fields in the message being replied-to, and a few switches given by the user, using the reply template, *repl* generates the reply draft automatically.

This facility, for example, can be used to generate automatic replies.[6] One function might be to write a *rcvtrip* shell script which automatically answered messages when mail wasn't being read for a period of time (e.g., while attending a conference). An example of a reply template at the heart of such a script is shown in Figure 4.

Finally, another application might be to utilize the highly useful letter bomb protocol.[7] The important thing to note about this template is that it generates not only the headers of the reply draft (with a creative ``Reply-to:'' address), but the body as well. Hence, the commands

```
repl␣-form␣bombcomps␣-noedit␣;␣rmm
```

[6] MH supports the notion of a user-defined *mail hook* which is invoked each time a user receives mail.
[7] The authors wish to credit Ron Natalie of the Ballistics Research Laboratory in Aberdeen, Maryland for formalizing the use of this protocol in the ARPA Internet community.

```
To: <reply-to|from>
cc: <?to|cc><to>,<cc>
Fcc: +outbox
Fcc: <?fcc><fcc>
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>.
Reply-To: /dev/null
--------

                          \
                         *-XXX
                          /   XX
                                X
                                  X
                                   X
                                   X
                                   X
                               IIIIIIIII
                               IIIIIIIII
                               IIIIIIIII
                               IIIIIIIII
                               XXXXXXXXX
                        XXXXXXXXXXXXXXXXXXXXXXX
                     XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
               XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
              XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
            XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
          XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
         XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
         XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
         XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
          XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
            XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
              XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
               XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                     XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                        XXXXXXXXXXXXXXXXXXXXXXX
                               XXXXXXXXX
```

Figure 5
The *bombcomps* Reply Template

```
What␣now?␣push
```

```
width=80,length=0,overflowtext=,overflowoffset=10
Date:leftadjust,compress,compwidth=9
Subject:leftadjust,compress,compwidth=9
From:leftadjust,compress,compwidth=9
To:leftadjust,compress,compwidth=9
cc:leftadjust,compress,compwidth=9
Resent-Note:leftadjust,compwidth=9
:
body:nocomponent,overflowoffset=0
```

Figure 6
Display Template

are very handy for dealing with disturbing mail in a straight-forward manner. Of course, *repl* could be linked to *bomb* in the user's *bin/* directory and an appropriate line could be added to the user's MH profile, in order to further shorten type-in.

A variation on the reply template is the *display template*. A display template, as used by the *mhl* program, contains instructions on how to format a message. In addition to being used by *show*, et. al., the *forw* program can also use a display template to format each message being forwarded. Similarly, although *repl* uses a reply template to construct the draft being composed, it also may use a display template to format the body of the message being replied-to for enclosure in the reply. Furthermore, the *post* program may use a display template to format the body of a blind-carbon-copy. An example of a display template used for formatting forwarded messages is shown in Figure 6.

As with reply templates, display templates can offer a lot of functionality. For example, the one line display template:

```
body:nocomponent,overflowtext=,overflowoffset=0,width=10000
```

can be used to extract the body of a message, while ignoring the headers. Hence, if a *shar* archive arrived in the mail, a convenient way to unpack it, assuming the above display template was called *mhl.body*, would be:

```
show␣-form␣mhl.body␣|␣sh
```

The biggest win with display templates, of course, is that all those annoying header lines which mailers everywhere generate can be simply and easily filtered out.

Modularity versus Monolithicity

Since MH is a set of programs which perform separate tasks, as opposed to being a single, monolithic program, the power of the shell is used directly to aid in mail-handling. One powerful capability which this design achieves is the ability to extend the MH command set, by developing shell scripts which use the standard MH programs to accomplish complicated or specialized tasks.

For example, in the MH distribution there is a shell script called *mpick* (shown in Figure 7) which tries to locate all the messages which pertain to a given discussion, by looking at the `''Message-ID:''` and `''In-reply-to:''` headers, to find matching message-ids.[8]

Unfortunately, some parts of MH are somewhat monolithic. An example of this is the `What now?` prompt. There are only a few options at this prompt, and one cannot give a normal shell command. Some MH users seem to feel that more options should be added to the `What now?` prompt, such as an *insert-file* option. It was argued that just about any editor would allow you to insert a file, and another `What now?` option was not needed. These users persisted, however, so the problem was solved, by writing a trivial shell script "editor" (see Figure 8) which could be invoked by the *edit* option:

```
What now?␣edit␣append␣filename
```

A better interface at this point is really needed, however. One possibility is to simply pass any unrecognized commands on to a shell for interpretation, supplying the path name of the draft file as an argument. A solution which shows more promise is to give you a sub-shell *instead* of the `What now?` prompt, and setup certain envariables so that the MH commands would act upon the *draft* by default. For example, *show* with no `'msgs'` arguments would show the draft instead of the current message. This alternative has recently been implemented and is under testing.

The MH Distribution

The mh.5 distribution is now briefly described, both in terms of static configuration methods and dynamic tailoring. Appendix B describes the mechanics of receiving an mh.5 distribution.

---

[8] Note that the shell scripts included in the MH distribution are written for the *Bourne* shell, and have a ':' as the first character of the first line, so they will be portable to all versions of UNIX, not just those which support the Berkeley '`#!`' enhancement.

```
: 'mpick - relate messages /mtr'
PATH=:/bin:/usr/bin:/usr/ucb:/usr/local:/usr/local/lib/mh; export PATH
F="" M="" S=""

for A in $*
do
    case $A in
        -*)     S="$S $A" ;;

        +*|@*)  case $F in
                    "")  F=$A ;;
                    *)   echo "mpick: only one folder at a time" 1>&2
                         exit 1 ;;
                esac ;;

        *)      M="$M $A" ;;
    esac
done

S="$S -sequence hits -list -nozero"

if mark $F all -add -sequence hits;
    then mark $F all -delete -sequence hits;
    else exit 1;
fi

for A in ${M-cur}
do
    for C in `mhpath $F $A`
    do
        if [ -r $C ];
            then
                I=`mhl -form mhl.msgid $C`;
                case $I in
                    "")  echo "no message-id in message `basename $C`" 1>&2 ;;
                    *)   pick --in-reply-to "$I" $S ;;
                esac
            else
                echo "message $A doesn't exist" 1>&2; exit 1;
        fi
    done
done

exit 0
```

Figure 7

The *mpick* Script

Configurable MH

The MH distribution currently runs on a large number of different UNIX versions, ranging from MicroSoft XENIX to Berkeley 4.2BSD. All the code which

```
: 'append - stupid append editor for MH - /jlr'
case $# in
   1|2) case $# in
            1) F=$1; echo -n "Append file: " 1>&2; read A ;;
            2) F=$2; A=$1 ;;
        esac
        cat $A < /dev/null >> $F ;;
     *) echo "append: arg count" 1>&2 ; exit 1 ;;
esac
exit
```

Figure 8
The *append* Editor

```
bin     /usr/local
bboards on
editor  /usr/local/prompter
etc     /usr/local/lib/mh
mail    /usr/spool/mail
manuals local
mts     sendmail/smtp
news    off
options BSD42
options MHE NETWORK
options UCI
```

Figure 9
Sample MH Configuration File

is specific to a particular target environment is enabled via the C-preprocessor ``#ifdef'' mechanism, so compilation under different versions of UNIX is trivial. There are, however, a large number of compile-time options which may vary from site to site, so an automated configuration method was needed.

The MH-installer must create a configuration file, which contains a list of the compile-time options and the values which are desired for them. Compile-time options include the installation location for MH, what kind of message transport system is to be used, and the default editor for the installation. An example of such a configuration file is shown in Figure 9.

After creating this file (several examples are included in the distribution), the installer runs the *mhconfig* program, which customizes the *Makefiles* and some of the programs, for that site's particular installation. No hand-editing of any source code should be necessary, under normal circumstances.

```
mmdfldir:       /usr/spool/mail
mmdflfil:
mmdelim1:       \001\001\001\001\n
mmdelim2:       \001\001\001\001\n
mmailid:        0
lockstyle:      0
lockldir:

hostable:       /usr/local/lib/mh/hosts
servers:        localhost \01localnet
```

Figure 10
Sample MTS Tailor File

Interface to the Message Transport System

MH will run with a number of message transport systems, including SendMail, MMDF-II, and a small stand-alone system. One flexible method of posting mail is through an SMTP connection. There are a couple of major wins in using this configuration: First, none of the MH programs need to know where the interface programs to the message transport system are located, which makes them easier to move between systems. Second, mail can be posted on relay hosts, and the local host of an MH user may not need a message transport system at all (as alluded to in the preceeding discussion on the POP).

Those parts of MH which interact with the local message transport agent read additional tailoring information when they start.[9] This information includes the location of standard and alternate maildrops, maildrop delimiter strings, the locking directory and locking style, and other tailoring information specific for the particular message transport system in use (e.g., the default server search-list when mail is posted with the SMTP). In most cases, by using a tailor file, each site running a similar MH configuration is able to simply transfer MH binaries between hosts. An example of such a tailor file is shown in Figure 10.

A continuing question which is often raised is how intelligent should user agents (like MH and UCB *Mail*) be with respect to the environment in which they operate. At present, MH likes to determine the official hostnames for addresses when posting mail. Many argue that this is improper or unnecessary behavior for a user agent, and that the local message transport agent should handle these functions. Unfortunately, this implies that the message transport agent should munge headers when mail is posted to remove local host aliases and only permit address fields with fully-qualified addresses. Sadly, neither SendMail nor MMDF-II really gets this right (flames to */dev/null* please). The current MH maintainers believe that the resolution of host aliases to official names should be a well-supported interface with the local message transport agent. However, to

[9] This simple facility is based on a more extensive tailoring capability found in MMDF-II.

provide equal time to those who hold opposite views, MH supports a configuration option called ``DUMB'' which disables MH's attempts to resolve addresses into fully-qualified strings.

Concluding Remarks

While MH has undergone significant development since the original Rand release, the authors have tried to keep the fundamental concepts of MH unchanged. The authors have continually had to battle against well-meaning MH users who wanted to make MH more like other (less powerful) user agents. More and more "features" were often suggested for MH, usually at the expense of making MH less general, and more specific. In nearly all cases, the "features" which these users wanted were already present in MH in a slightly different form, or could be realized by simply writing a short shell script. A classic example is the repeated requests by one user to have *dist* take a list of messages rather than a single message and distribute each one of them in turn. A simple shell script which called *dist* repeatedly, perhaps with "canned" arguments so the user typed in addressing information only once, would easily meet this request.

A number of MH comands have a large number of options. When adding options, the authors have tried to make the options general, while still accomodating the requests of specific users. An example of a specific request which was implemented as a general feature is the ``Previous-Sequence'' profile entry (mentioned above). If you use this profile entry, every MH command is forced to write out *context* changes, making every command somewhat slower. Since only a few users wanted this capability, it was implemented in such a way that users who didn't want it, didn't have to pay the cost of slowing down every MH command.

MH has a powerful tailoring capability provided by the *.mh_profile*. Using profile entries, users may customize their own environment without affecting others. Novice users often take advantage of the MH-tailoring capabilities to try to make MH work similarly to other user agents they've used. This has the advantage of allowing them to quickly begin using MH to handle their mail. However, since these novice users don't take advantange of all the capabilities of MH, they frequently will complain about things they think can't be done with MH, or could be done "better" some other way. Fortunately, as these users become more experienced with both MH and UNIX, they can modify their environment to take better advantage of all of MH's capabilities. Novice MH users who see features lacking are encouraged to take a better look at what MH *can* do, instead of trying to make MH into something it isn't. This may sound rather inflammatory, but it would really be a much nicer world for us all if users of software systems would read the manual prior to asking questions.

For a moment, let's consider the evolution of one MH feature which has proved itself to be very useful. As users began employing MH to handle their mail, the number of messages that could be processed in a given amount of time increased greatly. As the volume of messages increased however, it became clear that some MH operations were too slow, in particular the interaction with the (slow) message transport system. To overcome this problem, the *push* option was added at the `What now?` prompt. Originally, this option was hidden from novice users and did little more than send the message in the background: any output generated by the background *send* process would be printed asyncronously on the terminal. If a message failed posting with the message transport system, it would simply be left in the *draft* file.

Gradually, other features were added to *push*. Since users wanted to be able to send more than one draft at a time, *push* was changed to optionally rename the draft file before posting it. (This is what the hidden '`-unique`' option does.) Having message transport system diagnostics written asyncronously on the user's terminal was annoying, so *push* was made to intercept these diagnostics, and mail the user a report containing them. Although the diagnostic report mailed back by *push* contains the name of the draft which failed, a useful added feature was the ability to have *push* include the failed draft as well. Eventually, the draft-folder mechanism was implemented to make handling multiple message drafts much easier.

TODO

There are, no doubt, a number of improvements which could be made to MH. At the present time, what further development should MH suffer? Although not by any means inclusive, here's a list:

1. Performance Enhancements
   Hardware gets faster all the time, but people always complain that software is too slow. Owing to its user interface style, MH is somewhat slower than monolithic programs like UCB *Mail*. It would be nice if MH could be tuned or accelerated somehow.

2. Port to System 5
   MH runs on 4.2BSD UNIX and Version 7 variants. It should not be difficult to port MH to a SYS5 environment. This should significantly increase the number of hosts on which MH can run. The authors, lacking a SYS5 machine (and experience with SYS5) to perform the port, are actively seeking a System 5 guru to attempt this feat.

3. Interface to the Network News
   Not all sites that run MH are in the ARPA Internet, and as such the UCI BBoards facility may not be of much use to them. A good MH interface to the network news would allow users on hosts with a news feed to employ the same interface for reading and sending both mail and news.

4. Programmed Instruction for Beginners
   The complexity of MH is often intimidating to new users. It would be nice to develop a set of *learn* lessons for those users who don't like *man* pages and non-interactive tutorials.

5. Message List Expansion
   At present, when a list of messages is given to an MH command, it expands the list and processes each message in numerical order rather than the order in which the messages were given (e.g., ``show␣2␣1'' *show*s message 1 and then message 2). It would be nice if MH processed messages in the order they were given.

6. Context Changes
   In nearly all cases, an MH command does not write out context changes until it is about to exit successfully. There is some controversy as to whether this is the correct behavior in all cases. Some argue that once an MH command has fully parsed its argument list, the context should be updated.

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).

[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).

[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).

[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).

[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.

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:

<u>Composing Mail</u>

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

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.

<u>Posting Mail</u>

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).

<u>Reading Mail</u>

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.

Appendix B
Distribution Mechanics

The mh.5 distribution is available in two forms:

1. Anonymous FTP
   If you can FTP to the ARPA Internet, use anonymous FTP to the ARPAnet host UDel-Huey [10.2.0.96] and retrieve the file portal/mh.5–tar. This is a *tar* image of size 2.1 MB (approximately).

2. 9–track tape, 1600 bpi, *tar* format
   Otherwise, you can send $50.00 to the address below. This covers the cost of a magtape, handling, and shipping. In addition, you'll get a laser-printed hard-copy of the MH documentation. The documentation includes installation guide, MH Tutorial, MH User's Manual, changes document (from mh.4 to mh.5), and BBoards Manual.

   If you go with this option, be sure to include your USPS address with your check. Checks should be made payable to

   Regents of the University of California

   It's also a good idea (though not mandatory) to send a computer mail message to `Bug-MH@UCI` when you send your check via USPS to ensure minimal turn-around time. The distribution address is:

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

   714/856–6852

   Sadly, if you just want the hard-copies of the documentation, you still have to pay the $50.00. The *tar* image has the documentation source (the man is in ROFF format, but the rest are in TEX format).

In addition, there is some hope that mh.5, or a successor, might be found in a future 4.x Berkeley distribution.

Although MH is not "supported" per se, it does have a bug reporting address. Normally, the address `Bug-MH@UCI` is used to report bugs and bug fixes. There are however, two discussion groups which concern themselves with MH:

1. MH-Users@UCI
   A discussion group for the MH user community at large. Appropriate topics include: questions about how to use MH, tips on MH usage, and exchange of MH shell scripts. All requests to be added to or deleted from this list, along with problems, questions and suggestions, should be sent to `MH-Users-Request@UCI`.

2. MH-Workers@UCI
   A discussion group for MH maintainers and experts. Appropriate topics include: questions on how to configure MH, tips on MH configuration, exchange of MH bug reports (and fixes). All requests to be added to or deleted from this list, along with problems, questions and suggestions, should be sent to `MH-Workers-Request@UCI`.

The `''UCI''` host is also known as `''ucivax''`, so a possible UUCP path might be ... `!ucbvax!ucivax!bug-mh`.

Updates to MH are published on the `MH-Workers` list in the form of context diffs, and the appropriate distribution images are updated as well.

Contents

This document (version #1.43) was TEXset April 12, 1990 with DISS.STY v103.