Chapter 4 – Using the Mail System and the Editors

Reading Mail

This section explains how to read mail, how to examine and set mail options that pertain to reading mail, and *message sequencing*, which is a way of specifying that you wish to read or see a list of mail messages that meet certain criteria that you specify. Sending mail is addressed in the section that begins on page 57.

```
You have new mail (1 message).  Type 'help mail' for info on
reading it.
```

The simplest way to read mail that someone has sent you is to type:

```
@next on me
```

and to keep doing that until the system responds, `You have no next message.`

The MOO supports a generalized concept of a *mail recipient*, which is any object that can receive mail. The two categories of mail recipient are players and mailing lists. Each stores a collection of mail messages, and the commands for reading mail on yourself and on a mailing list are the same. Mailing list names are preceded by the asterisk character (`*`). In offline discussions, mailing lists are sometimes referred to as "star-lists".

When you log on, the MOO will tell you how many new messages you have (if any), and how many new messages are on *lists that you subscribe to. This implies two things: one, that there is a concept of subscribing to *lists, and two, that the system keeps track of which messages on a *list you've read and which you haven't. This information – what lists you're subscribed to and how far you've read in them – is stored on your player object in the `.current_message` property. Only you and wizards can examine this property, which means that you have some privacy concerning what *lists you choose to read. However, *lists themselves can record the fact that you read messages on them (or even @peek at messages on them), and that information is then available to the owner of the *list if e chooses to access it.

To start reading mail on a *list, you use the same command described above for reading mail on yourself, except that instead of typing `@next on me`, you'll type:

```
@next on <*list>
```

Note that reading any mail message on a list subscribes you to it. If you want to read all or part of a *list without subscribing to it, use the `@peek` command (see page 55).

When you first get your MOO character, you will not be subscribed to any lists. To see what mailing lists exist that you might subscribe to, type:

```
    @unsubscribed
```
[14]

or:

```
    @subscribe
```

To subscribe to a *list, type:

```
    @subscribe <*list>
```

for example, `@subscribe *News`. Or read a message on it. To read the first message on a list to which you aren't subscribed, type:

```
    @next on *list
```

To see what mailing lists you're currently subscribed to, type:

```
    @subscribed
```

You can get a sense of a *list's intended topic by reading its description. The system is able to recognize mailing lists by the asterisk, so you don't need to know a list's number to do this:

```
    look *News
    look *Chatter
```

If you've been logged on for a while and want to see if there are any new posts on lists you're subscribed to, you can type:

```
    @rn
```

An extremely useful command is:

```
    @nn
```

(Think "next new".) This command will read the next new message on yourself or on any *list you are subscribed to. Unlike `@next`, if it comes to the last new message on one list, it will start with the first new message on your next list, and so on until all new messages on all lists have been read. An alternative, if you want to read all your new mail in one shot, is the command:

```
    @ranm
```

(Think "read all new mail".) This will display the contents of all new messages on all your *lists non-stop. At the end, you will be presented with a `yes/no` prompt asking whether you got it all. If you didn't (for example, if you were disconnected in the middle) the system won't update the list of messages you've read, so that the same ones will still appear as new/unread messages when you next log on.

Now let's take a more detailed look at the `@next` command, along with its counterpart, `@prev`, combining them with the concept of your *current folder* and your

[14] On LambdaMOO this is *very long*. You might want to use a logging facility (this would be provided by a client program) or, alternatively, a command called `@netsend-mail-catalog` on feature object #27325.

current message on each folder. A *folder* is another name for a mail recipient, either yourself or a *list. (Think of a folder with a set of messages in it.)

If you type `@next` or `@prev` with no other arguments (specifiers), e.g,

```
@next
@prev
```

the system will print the next (or previous) message from your current folder, and will update its record of your current message on that folder. Suppose you have five old messages (on yourself), numbered 1-5, and five new messages, numbered 6-10. (Your current folder is and will always be yourself unless you change it, using the `@mail-options` command, detailed below.) Your current message is 5 (the last one you read). If you type `@next`, you will see message 6 on yourself, and your current message will be set to 6. If you type `@next` again, you will see message 7 on yourself, and your current message will be 7. If you type `@prev`, the system will print message 6, and will reset your current message to 6, and so on.

Now suppose you decide to read some messages on a *list. For the sake of illustration, we'll call it *chatter. You would type:

```
@next on *chatter
```

and the system would print the next message on `*chatter`, where "next" means "the next message that you haven't read yet", or, to put it another way, the next message after your current message on that folder. Note that your next message on that folder might be different from my next message on that folder, because you and I might not have read up to the same point. To continue reading messages on `*chatter`, you would type `@next on *chatter` repeatedly until you either came to the end of the list or had read as many messages as you wanted to.

You can also use `@next` and `@prev` to read more than one message at a time, either on yourself or on another list:

```
@next 3
@next 3 on *chatter
@next <number> on <*list>
```

You can also tell the system to treat the folder on which you most recently read a message as your current folder, instead of your current folder always being yourself. To do this, you would set one of your mail options using the `@mail-options` package. Options packages are simply groups of settings that you can use to customize your MOOing experience so that certain system behaviors are more to your liking. Suppose that you prefer not to have to type `@next on *chatter` each time, but would rather access the next message on that folder with a plain `@next` command with no arguments. To do this, type:

```
@mail-option +sticky
```

This makes the pointer to your current folder "stick" to the folder you last read a message on, rather than always reverting to yourself, and all mail commands will apply to that folder until you specify a different one. Some people find this more convenient. If you try this option and don't care for it, you can reverse it by typing:

```
@mail-option -sticky
```

Besides reading messages, you can also list message headers. A message header consists of a message number, the date it was sent, who sent it, and either its subject heading or the first few words of the first line (if there is no subject heading). One reason for listing message headers is to take advantage of the fact that you are not stuck just reading the next message on every list you're subscribed to. Rather, you can read messages selectively, either singly or in groups, if you so choose. Of course, you can list message headers selectively, too. This ability is a powerful tool when you want to search a list for a particular message, set of messages, or kind of message.

@mail

The `@mail` command shows you a list of messages, with summary headers. It can be used by itself, with no arguments, but it is an especially powerful searching tool when used with a specifier called a *message sequence*, which is a word or set of words that specifies a set of criteria that a message may or may not satisfy.

`@mail` typed by itself will show you the summary headers of the last 15 mail messages (you can change the number, with another mail-option) on your current mailing list. You could also type `@mail on *chatter` to peruse the last 15 posts on that mailing list, and so on.

Two typical uses for `@mail` are to preview messages on a list before reading them, or to try to identify a particular post that you want to re-read, respond to, or cite.

There are many useful message sequences that you can use to limit or shape the output you get. Here are a few examples:

`@mail new on me`	lists new messages for you
`@mail 43 on *think-tank`	lists message 43 on `*think-tank`
`@mail 43-53 on *think-tank`	lists messages 43-53 (inclusive) on `*think-tank`
`@mail next3 on me`	lists the next three messages on you (note, no colon or space between `next` and the number)
`@mail prev4 on me`	lists the previous 4 messages on you
`@mail first:5 on *theme`	lists the first 5 messages on `*theme`
`@mail last:6 on *B:Shutdown`	lists the last 6 messages on `*B:Shutdown`
`@mail 1-last on *newbies` `@mail 1-$ on *newbies`	lists all the messages on `*newbies` (The $ sign stands for "last".)
`@mail cur-$ on *geography`	lists messages headers from your current message through and including the last message on `*geography`

The above are based mainly on message numbers, your current folder, and your current message in a given folder. Here are some others that are even more flexible:

`@mail cur`	lists your current message on your current *list
`@mail from:Tower on *Research`	lists messages that Tower has sent to `*Research`
`@mail subj:infrastructure`	lists messages on your current *list with "infrastructure" in the subject heading
`@mail before:21-Mar`	lists messages on your current folder that were sent before March 21
`@mail until:30-Jun`	lists messages on your current folder that were sent on or before June 30
`@mail after:04-Jul`	lists messages on your current folder sent after July 4
`@mail since:31-Oct`	lists messages on your current folder sent on or after October 31
`@mail since:01-Dec body:unthemely on *Public-ARB`	lists messages on `*Public-ARB` posted since December 1 that have "unthemely" in the body of the message. (Note, body searches take a long time and should have an additional qualifier to narrow the search.)

If you do not specify a folder when using the `@mail` command, you will see the specified message headers on your current folder. If you do specify a folder (*list), you will see the specified message headers from that list (and if your mail options are set to +sticky, your current folder will be set to that *list).

See also `help mail` and `help message-sequences`.

@read, @peek

The `@read` and `@peek` commands display the full text of one or more messages rather than just the headers. They can take the same message sequence specifiers that the `@mail` command does. So, for example, you might do:

```
@read new on me
@peek last:5 on *think-tank
```

The differences between `@read` and `@peek` are these:

- `@read` subscribes you to the specified *list if you were not already subscribed. `@peek` does not.
- `@read` sets your current folder and current message to the last message read. `@peek` does not.
- `@read` suppresses the message if you have refused flames from the sender. `@peek` does not. (See `help @refuse.`)

@rmm

The `@rmm` command (think ReMove Message) removes one or more messages. It takes the same message sequence specifiers as `@mail`, `@read` and `@peek`. You can remove messages from yourself. You can remove messages that you have sent to a public *list (unless its owner has specified to the contrary) and you can remove messages from *lists that you own.

The preposition for the `@rmm` command is different: Whereas you `@read <message-sequence> ON <folder>`, you `@rmm <message-sequence> FROM <folder>`.

Messages that you have removed using the `@rmm` command aren't *really* gone until you either log out or `@rmm` some other messages, and you can "unremove" them with the `@unrmm` command. While they are in this indeterminate state, they are called *zombie messages.*

See also `help @unrmmail` and `help zombie-messages`.

@skip

Skip to the end of your current list. A good example of when you might choose to use this command is when you've been away from the MOO for a while and have a lots and lots of new messages on a *list. Some lists you may want to read all of, but with others you may just want to skip to the end without reading all the intervening messages.

@renumber

Some people like to renumber their messages from time to time, while others never do. When you remove a message (with the `@rmm` command), its number is not re-used. If you were to remove a message and subsequently list your messages, you would see a gap. Renumbering messages assigns new numbers to the messages (starting with 1) and removes the gap. So, if your current messages were numbered 1, 3, 5 and 17, and you typed `@renumber`, the same set of messages would now be numbered 1-4. It's a matter of taste, mostly. Note, though, that while it is fine to `@renumber` yourself, it can cause problems if you `@renumber` a *list that you own. The reason is that the record of everyone's current message on that list is kept not on the list, but on the players who subscribe to that list. If you `@renumber` a *list, then the `@rn` command might indicate to a player that the list has new mail, but the `@nn` command would say that there are no new messages. This is hardly catastrophic, but if you own a *list, it's better to let gaps in the message numbering stay.

*news

The `*news` mailing list is a regular list, like any other, except that only wizards can send to it, and wizards can cause designated messages to appear when a player types the command `news`. But you can read it like a regular mailing list. This would come in handy, for example, if you wanted to review an old `*news` message that wasn't actually appearing in the newspaper any more (because a wizard had removed its current-news designation).

Sending Mail (and getting a start on using the in-MOO editors)

It's good to get comfortable with the mail editor, because then you can send mail to individuals and compose posts to mailing lists and focus on what you have to say (write) rather than on the mechanics of writing it. Furthermore, the note editor and the verb editor work almost identically, so you can apply your mail editor skills later if you choose to program your own objects.

Practice is key. But when people practice, especially at the beginning, they make mistakes, and making mistakes feels embarrassing (to many people), and so they don't practice. My proposed remedy for this is to have you start by sending mail to *yourself* until you feel ready to go public, so to speak.

A bit of reassurance: You can't break the editor. If you get *completely* stuck, just log off – exit your client application or simply disconnect. In a minute or two, you'll be moved from the editor back to your home, and no harm has been done. So relax.

Beginning and Ending a Mail Session

A normal mail editing session begins with `@send` and ends with `send`. The difference is in the `@`-sign. If you change your mind about sending, you can type `abort` and your editing session will be thrown away. You can `@send` to a person, to a mailing list, or to any combination. Here are a few examples:

```
@send Yib
@send Yib Tower Drippy lovecraft Veren skeptopotamus
@send *Think-Tank
```

```
@send Yib *Think-Tank
@send me
```

(The last of these is what you would use to send mail to yourself.)

There is no case-sensitivity here, as in most things within the MOO except for passwords.

After beginning with `@send`, you will be prompted for a subject line. You may leave it blank if you wish by typing the `<enter>` key.

Once in the editor, you can type `look` at any time to see a list of available commands. Basic commands fall into three categories: adding text, displaying text, and changing text.

Adding Text

There are two basic ways to add text. In the editing rooms, what you say (with the `say` command) gets added to the text you are editing. That's one way. The other way is to type *the word*

```
enter
```

on a line by itself, followed by any number of lines of text, followed by a period on a line by itself. You can paste in lines of text from your computer's clipboard, this way, if you want to. This is the method that I prefer, but it's strictly a matter of taste. NOTE: Most client programs will wrap words to fit a reader's screen width, and trying to do that manually makes the text funny-looking for some people, and some of those will complain. As a point of style, you should type each paragraph as if it were one very long line, and separate your paragraphs with a blank line (by pressing the <enter> key twice in a row).

Displaying Text

After you've entered some text, you'll probably want to take a look at it and see whether you like it, or whether you want to adjust it. You can do either of the following:

```
print
```

or:

```
list
```

The first, `print`, displays all the text as it would be seen by the recipient. This is especially good for proofreading. The second, `list`, shows you the lines with their numbers. This is helpful because when making changes, you need to specify a line or lines by number. If the post is long, `list` may only print out some of the lines. You can force it to list all the lines by typing:

```
list 1-$
```

In this context the `$`-sign is a symbol that means “the end”. You can list any range of numbers.

Changing Text

To change something, use the `subst` command (think “substitute”), which can also be abbreviated as `s`:

```
subst /kind of fun/a whole lot of fun/5
subst /pretty good/fabulous/1-$
s /so-so/great/3-16
```

The first form will make the substitution in line 5, the second will make the substitution in all lines, the third in lines 3 through 16. The “`/`” character is a separator. It can actually be any character that doesn’t appear in the text you’re substituting.

To delete one or more lines, use the `delete` command (`del` for short):

```
delete 5
del 3-8
del 1-$
```

The editor will print out the line or lines deleted, for verification. Heads up: When you delete a line, the editor will move the insertion point to that line. This means that if you `enter` text again, it will appear *in place of* the line or lines you just deleted. To enter text elsewhere, use the `insert` command (`ins` for short):

`insert 1`	Get ready to insert text before line 1.
`insert 5`	Get ready to insert text before line 5.
`ins $`	Get ready to insert text at the end.

After the `insert` command, you must still type enter to actually add text to your message. Again, type a period on a line by itself when you’re finished entering lines.

Send it Off!

Type `send` to send the mail, or `abort` if you change your mind.

Mail Options

You can customize many aspects of how your mail messages are displayed and handled through an interface called an *options package*, which is simply a group of settings plus some commands to view and alter those settings. In the section on reading mail, you were shown `@mail-option +sticky`. In this section, I’ll explain what each mail option means. You can type `help @mail-options` go get a brief reminder of this section.

To list all your current mail options, type:

```
@mail-options
```

You will see something like this:

```
Current mail options:

 -include           Original message will not be included in
                    replies.
 -all               Replies will go to original sender only.
 -followup          No special reply action for messages with
                    non-player recipients.
 -nosubject         Mail editor will initially require a
                    subject line.
 -expert            Novice mail user.
 -enter             Mail editor will not start with an implicit
                    'enter' command.
 -sticky            Teflon folders:  mail commands always
                    default to 'on me'.
 -@mail             Default message sequence for @mail:
                    last:15
 -manymsgs          Willing to be spammed with arbitrarily many
                    messages/headers
 -replyto           No default Reply-to: field
 -netmail           Receive MOO-mail here on the MOO
 -expire            Unkept messages expire in 30 days (default)
 -resend_forw       @resend puts player in Resent-By: header
 -rn_order          .current_message folders are sorted by last
                    read date.
 -no_auto_forward   @netforward when expiring messages
 -expert_netfwd     @netforward confirms before emailing
                    messages
 -news              the 'news' command will show all news
 -no_dupcc          I want to read mail to me also sent to
                    lists I read
 -no_unsend         People may @unsend unread messages they
                    send to me
 -@unsend           Default message sequence for @unsend:
                    last:1
```

The items in the first column are the options themselves, which you can set. Most options are either *on* or *off*. In the example above, every option is preceded by a `-`, so all options are turned off. A few must be set to one of a selection of permissible text values. The syntax for setting (turning on) or resetting (turning off) options is as follows:

```
@mail-option +<flag>
@mail-option -<flag>
@mail-option !<flag>
```

(The last two are equivilent.)

OR (if required):

```
@mail-option <option>=<value>
```

Let's go through them one by one.

```
@mail-option +include
@mail-option -include
```

Sooner or later you are bound to see a mail message that looks something like this:

```
Message 122 on *Features (#42343):
Date:    Tue Jan 18 17:41:37 2000 PST
From:    Yib (#58337)
To:      *Features (#42343)
Subject: Re: @parties

>  Date:     Tue Jan 18 14:24:01 2000 PST
>  From:     Goldmund (#96860)
>  To:       *Features (#42343)
>  Subject:  @parties
>
>  Good work Yib.
>  It would be nice if the rooms would be sorted by
>  the number of occupants rather than by alphabet.
>  ... and you should add it to the @help.
>
>  thanks

Done, and done.  Thanks for the suggestions.
--------------------------
```

Embedded in this message is a prior message, each line of which is preceded by the ">" symbol. Sooner or later you'll wonder, "How does she do that?" The `@reply` command does this automatically if the `include` mail option is turned on. Goldmund's message was message 121 on *features. When I typed `@reply 121 on *features`, the mail editor automatically created the header and included the text of the message I was replying to. I then appended my own lines of text.

Note: I also typed `to *features` to send my post to the mailing list as part of an ongoing discussion (in this case, discussion of a feature verb I'd written) instead of only to Goldmund. The `@reply` command directs the reply to the author of the post you're replying to – and not the list you read it on – *unless* you have either the `all` option or the `followup` option turned on.

```
@mail-option +all
@mail-option -all
```

If I had set the `+all` mail option, then my reply would automatically have been to Goldmund *and* `*features`.

```
@mail-option +followup
@mail-option -followup
```

It gets better. If the `+followup` option is set, and I `@reply` to a message, the message will be to "the first non-player recipient, if any". What's a non-player recipient? A mailing list. What if you're replying to a private message from a player and not a message on a mailing list (i.e. what if there *are* no non-player recipients)? Then it uses whatever you've set the `all` flag to, either the original author (`-all`) or all the original recipients (`+all`).

Reminder: You can override the designated recipients from within the mail editor at any time with the `to` command. `To` Goldmund would direct a post just to Goldmund. `To` *features would direct it just to *features.

```
@mail-option -nosubject
@mail-option +nosubject
```

When you type the `@send` command, the system prompts you to enter a subject line, unless you have the `+nosubject` option set.

```
@mail-option +expert
@mail-option -expert
```

The online documentation tells you that the `+expert` mail option suppresses "varying annoying messages". What are these various annoying messages? It turns out that there's only one. If you are `-expert` and receive mail, you will see something like this:

```
You have new mail (22) from Yib (#58337).
Type 'help mail' for info on reading it.
```

If you are `+expert`, you won't see the line that says, `Type 'help mail' for info on reading it.` Rog (the author of this code) got tired of seeing this message all the time, and so added this option to suppress it plus any other messages he decided to add later. No other messages suggested themselves, so that's all, folks.

```
@mail-option +sticky
@mail-option -sticky
```

This option was described in the segment on reading mail. It affects the `@mail`, `@read`, `@peek`, `@next`, `@prev`, and `@answer` (same as `@reply`) commands. If the option is set to `+sticky`, any of the above commands will apply by default to the mail recipient (yourself or a *list) you accessed most recently. If its set to `-sticky`, your current mail recipient (current folder) will always be yourself.

```
@mail-option +netmail
@mail-option -netmail
```

You can choose whether to receive your MOOmail in-MOO (`-netmail`), or have your MOOmail automatically forwarded to your registration email address (`+netmail`). The latter saves space (for the MOO), but the trade-off is that if you receive mail while logged on, you don't get to read it until it reaches your regular email address, and for some people that's a hassle.

```
@mail-option +resend_forw
@mail-option -resend_forw
```

You can either `@forward` or `@resend` a message (on yourself or a *list) to another player or another *list. These two commands are awfully similar, differing only in what the header information looks like, possibly modified by the `resend_forw` mail option. Here are some examples to show you what I mean: I started by sending a message to myself, then forwarded that message to another character, Yib's_Assistant.

Here is the original message that I sent to myself:

```
Date:     Wed Jan 19 16:28:08 2000 PST
From:     Yib (#58337)
To:       Yib (#58337)
Subject:  Secret

La plume de ma tante est dans le jardin.
--------------------------
```

When you `@forward` this message, you get:

```
Message 7:
Date:    Wed Jan 19 16:28:30 2000 PST
From:    Yib (#58337)
To:      Yib's_Assistant (#61050)
Subject: [Yib (#58337):  Test]

Date:     Wed Jan 19 16:28:08 2000 PST
From:     Yib (#58337)
To:       Yib (#58337)
Subject:  Secret

La plume de ma tante est dans le jardin.
--------------------------
```

Notice that there are two header blocks, and brackets around the original subject line. If you `@resend` the message, with `-resend_forw`, you get only one header block:

```
Message 8:
Date:    Wed Jan 19 16:28:38 2000 PST
From:    Yib (#58337)
To:      Yib (#58337)
Subject: Secret
Resent-By:     Yib (#58337)
Resent-To:     Yib's_Assistant (#61050)
Original-Date: Wed Jan 19 16:28:08 2000 PST

La plume de ma tante est dans le jardin.
--------------------------
```

`@resend` with `+resend_forw` generates just one header, but this time there are two “Original” lines instead of “Resent-” lines:

```
Message 9:
Date:    Wed Jan 19 16:28:56 2000 PST
From:    Yib (#58337)
To:      Yib's_Assistant (#61050)
Subject: Secret
Original-Date: Wed Jan 19 16:28:08 2000 PST
Original-From: Yib (#58337)

La plume de ma tante est dans le jardin.
--------------------------
```

```
@mail-option +no_auto_forward
@mail-option -no_auto_forward
```

Mail messages are one of the bigger database hogs on a MOO. Therefore, on some MOOs, mail is automatically deleted from players and *lists after a certain amount of time (usually 30 days). By default, when your messages expire, they are forwarded to your registration email address. Setting this option inhibits that forwarding.

```
@mail-option +expert_netfwd
@mail-option -expert_netfwd
```

You can forward MOOmail messages on yourself or on *lists to your registration email address with the `@netforward` command. By default, the system will display your current registration email address and prompt you for confirmation. Setting this option inhibits that.

```
@mail-option manymsgs is <number>
@mail-option manymsgs <number>
@mail-option manymsgs=<number>
@mail-option -manymsgs
```

Suppose you typed `@read new on *chatter` and suddenly found yourself spammed with 300 messages! The above option gives you some control over that kind of situation. If you specify a number for the `manymsgs` option, you will be given a yes/no prompt to continue. The last form, -manymsgs, inhibits this behavior.

```
      @mail-option @mail <message-sequence>
      @mail-option @mail is <message-sequence>
```

For example:

```
      @mail-options @mail=new
```

On most MOOs, typing the command @mail is equivalent to typing @mail last:15 to show the last 15 messages on your current folder. There are other possibilities, such as @mail new or @mail 1-last, and you can specify how you want plain @mail to behave by using this option.

```
      @mail-option replyto <recipient> [<recipient>...]
      @mail-option replyto is <recipient> [<recipient>...]
      @mail-option -replyto
```

The above option automatically inserts a "Reply to:" field in any messages that you compose using @send or @reply to indicate that replies should be sent to someone other than the original sender. The last form resets this option so that no "Reply to: " field is initially inserted.

```
      @mail-option rn_order=<order>
      @mail-option rn_order <order>
```

This option controls the order in which folders will be listed when you use the @rn and @subscribed commands. The options for <order> are:

read	Folders are sorted by last read date. This is the default.
send	Folders are sorted by the date you last sent to them.
fixed	Folders are not sorted initially.

If you specify rn_order=fixed, you can sort your folders using the @subscribe command, and they will stay in that order afterwards. For example, you can type @subscribe *random after *life, or @subscribe *random before *life. By doing this several times, you can re-order all your folders.

```
      @mail-option expire <time-interval>
      @mail-option expire is <time-interval>
      @mail-option expire=<time-interval>
```

By default, mail messages expire after a certain time period (usually 30 days). You can use the above option to change this. <time-interval> is a number of seconds unless you specify units, e.g. 5 weeks or 2 months.

A negative number or:

```
      @mail-option +expire
```

disables expiration entirely. You can keep a particular mail message from expiring with the @keepmail command.

```
      @mail-option -expire
```

sets your message expiration time to MOO's current default.

```
      @mail-option +no_unsend
      @mail-option –no_unsend
```

LambdaMOO provides the ability to take back a MOOmail message sent to an individual if the message has not yet been read, and has not been netforwarded. Setting this option prevents people from being able to `@unsend` messages they have sent to you.

```
      @mail-option @unsend=<message sequence>
      @mail-option -@unsend
```

If you wish to retract a message you just sent to someone, you can try `@unsend from <player>`. If you don't specify which message(s) you wish to `@unsend`, the system will use `last:1`, meaning the last message you sent. You can't look at another person's mail headers to specify message numbers, but you can specify other parameters such as `subj:<part of a subject heading>` or `since:yesterday`, and so on. You can do this for a single invocation of `@unsend`, or you can set such a filter to be the default, by using this mail option.

Using the in-MOO Editors

There are three editors provided with LambdaCore: the mail editor, for editing and sending MOOmail, the verb editor, for editing and compiling verbs (MOOcode), and the note editor, for editing all other bodies of text. Each editor has a few unique commands, e.g. you `send` a message, `compile` a verb, and `save` other text, but other than that, they are fundamentally the same, and if you can use one, you can use the others.

Editors in the MOO are implemented as special rooms! These rooms simulate a player being alone (if other people are using the editor simultaneously, you will not see them), and in these rooms, the verbs `say` and `emote` are programmed to work differently – specifically, `say` and `emote` are ways of inserting or appending text to whatever you are editing.

These are line-based editors, which means that text is added a line at a time. There isn't a cursor, as there is in most screen editors. Instead, there is the concept of an *insertion point*, and this number represents where new text will be added: between two lines, before the first line, or after the last line. Lines can in fact be quite long – as long as a conventional paragraph, in fact, comprising many sentences. They may take up more than one horizontal row of text on a person's screen. A *line*, then, in this context, is a logical unit of text. Typical usage is to type in one entire paragraph of text per line in the editor, and use blank lines to separate paragraphs. Most people use some sort of client software that formats text to the width of their screen or window as needed, and the one paragraph per line method makes it easiest for most people's clients to present text in a coherent way.

I'll begin by talking about how to begin an editing session, and how to end one, i.e. how to get out of the editor once you're there. Then I will describe some different ways to enter text, ways to view the text you've entered so far, and ways to modify the text you've entered.

Invoking the Editor

When you start up one of the editors, you are moved to one of the special editing rooms, and we call that *invoking* that editor. Thus, one would *invoke* the note editor, the mail editor, or the verb editor. The in-MOO editors require you to specify what you're editing before you can begin. (This differs from most commercial editor or word processing programs which provide the option of opening a new file, then specifying where the text will go, for example with a "Save As" command.) So you begin editing by indicating the thing you want to edit. Here are some examples:

`@send Werebull`	Invoke the mail editor, working on a message to the player Werebull.
`@send *chatter`	Invoke the mail editor, working on a message to the list `*chatter.`
`@send Werebull *chatter`	Invoke the mail editor, working a message for both Werebull and `*chatter.`
`@notedit me.description`	Invoke the note editor, working on your own `.description` property.
`@edit me.description`	Invoke the note editor, working on your own `.description` property (same result as `@notedit me.description`).

`@notedit here.exterior_description`
 	
`@edit here.exterior_description`
	Invoke the note editor, working on the `.exterior_description` property of the room that you are in. Note that you have to own an object in order to edit properties on it.

`@notedit test note`
 	
`@edit test note`
	Invoke the note editor, working on the text of an object named "`test note`".

`@notedit #1234`
 	
`@edit #1234`
	Invoke the note editor, working on the description of object `#1234` (which you must own).

`@edit rock:drop`	Invoke the verb editor, working on the verb `:drop` on an object named “`rock`”.
`@edit #1234:drop`	Invoke the verb editor, working on the verb `:drop` on object #1234

Notice that the same command, `@edit`, can invoke either the note editor or the verb editor, depending on whether you use a period (`.`) to specify a property, or a colon (`:`), which indicates a verb. Furthermore, if you `@notedit` or `@edit` a descendent of either the generic note or the generic letter, you will be editing that note or letter’s text, but if you `@notedit` or `@edit` any other object (without specifying a property or verb), then you will edit that object’s description.

```
@edit #1234
```

will invoke the note editor to edit the *description* of object #1234.

Leaving an Editor

There are three ways that you can leave an editor after you have finished entering and editing text.

One is to exit and discard all text you may have entered or modified during that editing session. The command is the same in all three editors. Type the command:

```
abort
```

The second is to save your changes or send your message and then exit, and this is different in each of the editors. In the note editor, type `save`, and then `done`. In the mail editor, type `send`. In the verb editor, type `compile` and then `done`.

The third way is to leave your work in progress. In any of the editors, you can type `done` (without saving, sending, or compiling). Or you can teleport from the editor (remember, an editor is a room) to any other location. Or you can log off. Your work is saved on the equivalent of a scratch pad, and when you return to the editor, you can either resume that work, or abort out of it. To return, you can type `@edit` by itself (for the note editor or the verb editor) or `@send` by itself (for the mail editor). If you try to edit something other than what you were working on, the system will ask you if you want to resume the original editing session or abort it, before letting you edit the new note verb, or MOOmail.

Listing Text, Finding and Moving the Insertion Point

When you enter new lines of text, they are added after the current line. So the first order of business might be to determine which line is current and perhaps move to another line. The place where new text will be added is called the *insertion point.*

The easiest way to see where the insertion point is, and get your bearings in the text, is to type:

```
list
```

This will show you the current insertion point, plus some of the context, i.e. a few of the lines above and a few below. Here's one example:

```
  1: Fee,
  2: Fie,
__3_ Fum!
^^^^
```

In this case, the insertion point is after line 3, and if I begin entering text, it will go in at the bottom of the document. If the insertion point were between lines 2 and 3, then `list` would show:

```
  1: Fee,
__2_ Fie,
^^3^ Fum!
```

The combination of underscores and up-arrows is intended to depict a spot between two lines, where new text will go, if and when you start entering it.

There are a few ways to move the insertion point. Perhaps the easiest is with the `insert` command, which can be abbreviated `ins`. Counter to what you might think, the `insert` command doesn't insert text. It gets you ready to insert text by specifying *where* you want to insert it. Used alone, the `insert` command will show you the insertion point with just the line above and the line below. If you specify a number, though, that moves the insertion point before that line:

`ins 1`	Get ready to insert text before line 1, i.e. at the very beginning of the document.
`ins _1`	Get ready to insert text after line 1, i.e. between lines 1 and 2.
`ins 3`	Get ready to insert text before line 3, i.e. between lines 2 and 3.
`ins $`	Get ready to insert text at the end of the document, i.e. after the last line.

Entering Text

There are four ways to input new text into your document, mail message, or verb. These are: `say`, `emote`, `enter`, and `yank`.

When you say something in an editor, the text you say is added as a new line at the insertion point. So if, in our example, the insertion point were between lines 2 and 3, and you typed:

```
"Foo,
```

and then you typed `list`, you would see:

```
  1: Fee,
  2: Fie,
__3_ Foo,
^^4^ Fum!
```

(Notice that the double-quote character (") didn't get added, because it is just the abbreviation for the say command.)

When you emote something in an editor, it appends that text to the end of the line that is *before* the insertion point. Suppose you wanted to append some text to line 4. First you would position the insertion point after line 4 by typing either `ins _4` or `ins $`. Then you might type:

```
:  I smell the blood of an Englishman!
```

and then if you typed `list`, you would see:

```
  1: Fee,
  2: Fie,
  3: Foo,
__4_ Fum!  I smell the blood of an Englishman!
^^^^
```

Another way to enter text is with the `enter` command. This entails typing the word `enter` on a line by itself. The system prompts you to enter lines of text. You terminate this text entry by typing a period (`.`) on a line by itself, or else the command `@abort`. The first keeps (and inserts) the lines of text, the second throws them away. Suppose you wanted to add some text to the beginning of the document. First, position the insertion point so that it is at the beginning: `ins 1`. Then type (exactly):

```
enter
The giant looked around...
Then the giant sniffed the air...
Then the giant began to roar:
.
```

Now if you type `list`, you will see:

```
  1: The giant looked around...
  2: Then the giant sniffed the air...
__3_ Then the giant began to roar:
^^4^ Fee,
  5: Fie,
  6: Foo,
  7: Fum!  I smell the blood of an Englishman!
```

You can also compose text in a word processing window on your own computer and paste it from the clipboard after you type the word `enter` and before you type the period on a line by itself.

The last way to enter text is with the `yank` command. It's used to bring in the text of a note object in its entirety. So if you had the entire story of Jack and the Beanstalk in a note object, and wanted to pull it into a mail message, for example, you would start the mail editor with the `@send <person>` command, then (after typing the subject line, if prompted to do so) you would type:

```
yank from <note>
```

The entire text from the specified note would be entered as if you had typed it. You could then edit it further, or save or send it as it was.

In addition to listing text, you can display it without the numbers, by typing the command:

```
print
```

Modifying Text

Once you have some text and looked it over, you might want to make some modifications. For this we have the `subst` command, which can be abbreviated as `s`. If you type `look` while in any of the editors, you'll see a short synopsis of commands with their syntaxes. The one for `subst` looks like this:

```
s*ubst      /<str1>/<str2>[/[g][c][<range>]]
```

The heart of the `subst` command is to substitute one string for another. The simplest version is to make the substitution in the current line. Let's list out our text, again:

```
  1: The giant looked around...
  2: Then the giant sniffed the air...
__3_ Then the giant began to roar:
^^4^ Fee,
  5: Fie,
  6: Foo,
  7: Fum!  I smell the blood of an Englishman!
```

The insertion point is after line 3, which makes line 3 the current line for substitution. Let's change "roar" to "bellow":

```
subst /roar/bellow/
```

The editor will print out the modified line:

```
__3_ Then the giant began to bellow:
```

Now let's change that "Foo" to "Fo" in line 6:

```
subst /Foo/Fo/6
```

The editor will again print out the modified line:

```
6: Fo,
```

Let's change the giant into an ogre, through the whole range of lines. Remember that the way to designate "after the last line" is with the "$" sign:

```
subst /giant/ogre/1-$
```

We will see:

```
  1: The ogre looked around...
  2: Then the ogre sniffed the air...
__3_ Then the ogre began to bellow:
```

By now you might want to take a look at the full text again, to see what we've got:

```
print
```

You will see:

```
The ogre looked around...
Then the ogre sniffed the air...
Then the ogre began to bellow:
Fee,
Fie,
Fo,
Fum!  I smell the blood of an Englishman!
--------------------------
```

For fun, let's change all instances of the letter "f" to the letter "r". This incorporates everything at once:

```
subst /f/r/gc 1-$
```

The "g" (think "global") in the command means every letter "f" on a line, not just the first, so we get "snirred" rather than "snirfed". The "c" means ignore capitalization. So we get:

```
The ogre looked around...
Then the ogre snirred the air...
Then the ogre began to bellow:
ree,
rie,
ro,
rum!  I smell the blood or an Englishman!
```

At this point, we've pretty well mangled our example. How to restore it? More substitutions. Or type `abort` and start over.

Before we leave substitutions, let's consider what to do if you wanted to use the slash character (`/`) itself as part of a substitution: You can use any separator at all that doesn't appear in either the string you wish to replace or the new string. So to change "and or" to "and/or" you could type any of:

```
subst |and or|and/or|
subst ,and or,and/or,
subst xand orxand/orx
```

The `subst` command is used so much that it finds its way into casual conversation. You might see something like this:

```
Ostrich says, "I will now sing the national anthem while
standing on my head and drinking root beer through a straw."
Ostrich says, "subst /now/not/"
```

Ostrich is indicating that he made a typographical error and intended to say that he will *not* sing the national anthem while standing on his head and drinking root beer through a straw.

Re-arranging Text

Let’s go back to an earlier version of our text:

```
  1: The giant looked around...
  2: Then the giant sniffed the air...
  3: Then the giant began to roar:
  4: Fee,
  5: Fie,
  6: Foo,
__7_ Fum!  I smell the blood of an Englishman!
^^^^
```

The `move` command moves a line of text, or a range of lines, to a new point. If we wanted to change the sequence of our giant’s utterances, for instance, we could do this:

```
move 6 to 5
```

and get this:

```
  1: The giant looked around...
  2: Then the giant sniffed the air...
  3: Then the giant began to roar:
  4: Fee,
  5: Foo,
  6: Fie,
__7_ Fum!  I smell the blood of an Englishman!
^^^^
```

We could do this:

```
move 5-6 to 4
```

and get this:

```
  1: The giant looked around...
  2: Then the giant sniffed the air...
  3: Then the giant began to roar:
  4: Foo,
  5: Fie,
  6: Fee,
    __7_ Fum!  I smell the blood of an Englishman!
    ^^^^
```

The `copy` command copies a line or range of lines in like manner:

```
    copy 4 to 4
```

would yield:

```
      1: The giant looked around...
      2: Then the giant sniffed the air...
      3: Then the giant began to roar:
      4: Foo,
      5: Foo,
      6: Fie,
      7: Fee,
    __8_ Fum!  I smell the blood of an Englishman!
    ^^^^
```

The `join` command joins lines together. For example:

```
    join 4-8
```

gives:

```
      1: The giant looked around...
      2: Then the giant sniffed the air...
      3: Then the giant began to roar:
    __4_ Foo, Foo, Fie, Fee, Fum!  I smell the blood of an
    Englishman!
    ^^^^
```

The `fill` command combines joining and splitting lines so that they are less than or equal to a specified number of characters (columns) in length. For example:

```
    fill 4 @25
```

gives:

```
      1: The giant looked around...
      2: Then the giant sniffed the air...
      3: Then the giant began to roar:
      4: Foo, Foo, Fie, Fee, Fum!
      5: I smell the blood of an
    __6_ Englishman!
    ^^^^
```

Deleting lines

You can delete a line or range of lines with the `delete` command, which can be abbreviated as `del`. When you do this, the editor prints out the deleted text *and* the insertion point is automatically moved to the point where the lines were deleted, so if you then start entering text, it will directly replace the lines you just deleted. Here's a quick example:

```
del 4
Foo, Foo, Fie, Fee, Fum!

"Fee, Fie, Fo, Fum!
Line 4 added.

list

  1: The giant looked around...
  2: Then the giant sniffed the air...
  3: Then the giant began to roar:
__4_ Fee, Fie, Fo, Fum!
^^5^ I smell the blood
  6: of an Englishman!
```

A Few Miscellaneous Commands

If you somehow forget what you are editing, you can use the `what` command. I edited an actual note on LambdaMOO as I wrote this segment. Here's what I see:

```
what

You are editing "test note"(#92413).
Your insertion point is before line 5.
There are changes.
```

Sometimes you want to verify a line number before doing a substitution, and for this we have the `find` command. By itself, it will find the next instance (i.e. after the current insertion point, wherever that is) of the string you specify. Or you can specify a starting point. You can also ignore capitalization. The `find` command moves the insertion point, so you can use the command sequentially until you find the particular instance of a word or string of characters that you're looking for. Here are some examples:

`find /f`	Find the next line with an “f” in it.
`find /f`	Find the next-next line with an “f” in it.
`find /f/1`	Find the first “f” in or after line 1.
`find /f/c1`	Find the first “F” or “f” in or after line 1.
`find /fo/c1`	Find the first instance of “fo” or “FO” or “Fo” or “fO” in or after line 1.

The `mode` command is probably of interest only to programmers. If you are not a programmer and don't understand this part, don't worry; you may ignore it in complete safety. If your text has several lines in it, it will be stored, when you save it, as a list of strings. If your text has only one line of text in it, then it can either be stored as a single text string *or* as a list with one element (a string) in it.

`mode`	Find out what mode you're currently in.
`mode string`	Switch to string mode.
`mode list`	Switch to list mode.

The `publish` and `view` commands work together. Normally, no one can see your text as you edit it. But suppose you were struggling with something in the verb editor, and I wanted to help you but couldn't quite get a handle on the situation based on paging each other back and forth. First I would go to a "blank" editing session by typing `@go $verb_editor` (remember, it's a room). Then I would page you to type the `publish` command. Then I could type `view <your name>` and the system would list out what was in your editing session. I might then page you with something like, "Try adding a semicolon to the end of line 3."

In the mail editor, you can change the recipient of your post with the `to` command. Suppose you had typed `@reply cur on *chatter`, and because of your mail-option settings, your post was addressed to the author when what you really wanted to do was reply on the *list itself. You'd spot this (one hopes) when printing your message before sending it. You could change the recipient by typing `to *chatter`, or `to <list of recipients>`.

Edit Options

Just as there are mail-options you can set, there are also edit-options you can set. Type `@edit-options` to see what they are. Probably the most significant one is `local`. If you type `@edit-option +local`, then whenever you `@edit` anything, the text you wish to edit will be uploaded to your client program, and when you are finished, you can download the text again. Each client is different, as are the commands to edit in those clients, and not all clients support this feature. To change back to using the in-MOO editors, type `@edit-option –local`.

Summary

If you forget the collection of commands available to you, you can type `look` while in any of the editors and see a reminder list of them. There is detailed help text for each individual command as well.

```
look

Note Editor
Commands:

say         <text>                      y*ank       from <text-source>
emote       <text>                      w*hat
lis*t       [<range>] [nonum]           mode        [string|list]
ins*ert     [<ins>] ["<text>]           e*dit       <note>
n*ext,p*rev [n] ["<text>]               save        [<note>]
enter                                   abort
del*ete     [<range>]                   q*uit,done,pause
f*ind       /<str>[/[c][<range>]]
s*ubst      /<str1>/<str2>[/[g][c][<range>]]
m*ove,c*opy [<range>] to <ins>
join*l      [<range>]
fill        [<range>] [@<col>]

----  Do `help <cmdname>' for help with a given command.  ----

<ins> ::= $ (the end) | [^]n (above line n) | _n (below line n) | . (current)
<range> ::= <lin> | <lin>-<lin> | from <lin> | to <lin> | from <lin> to <lin>
<lin> ::= n | [n]$ (n from the end) | [n]_ (n before .) | [n]^ (n after .)
‘help insert' and ‘help ranges' describe these in detail.
```