GNU Mailutils

version 1.0, 17 May 2005

Alain Magloire et al.

Published by the Free Software Foundation, 51 Franklin Street, Fifth Floor Boston, MA 02110-1301, USA

Short Contents

Table of Contents

1 Introduction

GNU Mailutils contains a series of useful mail clients, servers, and libraries. These are the primary mail utilities of the GNU system. Specifically, this package contains a POP3 server, an IMAP4 server, and a Sieve mail filter. It also provides a POSIX 'mailx' client, and a collection of other tools. The central library is capable of accessing different mailbox formats and mailers as well as off of local or remote POP3 and IMAP4 servers.

This software is part of the GNU Project and belongs to the Free Software Foundation. All libraries are licensed using the GNU LGPL. The documentation is licensed under the GNU FDL, and everything else is licensed using the GNU GPL.

Why use this package?

This package started off to try and handle large mailbox files more gracefully then current POP3 servers did. While it handles this task, it also allows you to support a variety of different mailbox formats without any real effort on your part. Also, if a new format is added at a later date, your program will support that new format automatically as soon as it is compiled against the new library.

2 Mailutils Programs

GNU Mailutils provides a set of programs for handling the e-mail.

2.1 Mailutils Configuration File

There are some command line options that are used so often that it is inconvenient to specify them in the command line each time you run a Mailutils utility. The *configuration files* provide a way to add default command line arguments without having to type them in the command line. Upon startup, each Mailutils utility scans and processes the contents of the three startup files, none of which are required to exist:

1. the site-wide configuration file

   '`mailutils.rc`', found in your your system configuration directory (usually '`/etc`' or '`/usr/local/etc`').

2. the user-specific configuration file

   Usually '`~/.mailutils`', unless '`~/.mailutils`' is a directory, in which case '`~/.mailutils/mailutils`' is used.

3. the programs-specific configuration file

   Usually '`~/.mu.`*program*`rc`', unless '`~/.mailutils`' is a directory, in which case '`~/.mailutils/`*program*`rc`' is used (where *program* means the program name).

These files have simple line-oriented syntax. Comments begin with the pound sign ('`#`') and extend through the end of the line[1]. Very long lines may be split across several lines by escaping final newline with a backslash ('\') character.

In the non-program-specific configuration files, any configuration line must start with a *tag*. In the program-specific configuration file the tag must not be present, all options are for that specific program.

A tag is either a name of a particular Mailutils utility or *option group*, prefixed with colon ('`:`'). The command line options common for several Mailutils programs are divided into *option groups* or *capabilities*, e.g. the options '`--mail-spool`' and '`--lock-flags`' form group '`mailbox`'. These groups are discussed in detail below.

When processing the non-program-specific configuration files, a Mailutils utility selects those lines whose tag is either the name of that utility or the name of the option group supported by it. In the program-specific configuration file, all lines are selected. For each line found, its tag (if present) is stripped away, and the rest of the line is split up into words. These words are regarded as command line options and are inserted to the program arguments *before* any options from the command line. Thus the options from '`.mailutils`' take precedence over those from '`mailutils.rc`', and the options from the command line take precedence over those from all three configuration files.

The word splitting occurs at whitespace characters and is similar to that performed by the shell. If an option must contain embedded whitespace, it should be enclosed in a pair of quotes (either double or single quotes).

[1] If '`#`' is not the first character on the line, it should be separated from the previous word by any amount of whitespace.

2.1.1 Common

Each program understands the following informational options:

‘`-u`’
‘`--usage`’ Display a short usage message and exit.

‘`-h`’
‘`--help`’ Display help message and exit.

‘`-L`’
‘`--license`’
Display GNU General Public License and exit.

‘`-v`’
‘`--version`’
Display program version and exit.

2.1.2 Mailbox

Option group ‘`mailbox`’ consists of options used to specify the location of the mail spool, and the locking strategy.

‘`-m` *`path`*’
‘`--mail-spool=`*`path`*’
Set path to the mailspool directory

‘`--lock-flags=`*`flags`*’
Set the default mailbox lock flags (E=external, R=retry, T=time, P=pid).

2.1.3 Mailer

This option group overrides the default mailer URL (`sendmail:`).

‘`-m` *`url`*’
‘`--mailer` *`url`*’

2.1.4 Address

Option group ‘`address`’ consists of options used to specify how to qualify email addresses.

An unqualified address (one without an `@`) is qualified by appending `@`*defaultdomain*. *defaultdomain* is the return of `gethostname()`, or the result of `gethostbyname()` on the return of `gethostname()` (if the DNS lookup is successful).

If the email address of the current user is needed, either the address set by ‘`--email-addr`’ is returned, or the current uid is looked up in the user database, and qualified with the *defaultdomain*.

‘`-E` *`email`*’
‘`--email-addr=`*`email`*’
Set the current user’s email address, this it makes more sense to use this in one of the per-user configuration files.

‘`-D` *`domain`*’
‘`--email-domain=`*`domain`*’
Set the default email domain, this might make sense to use in either the global or one of the per-user configuration files.

2.1.5 Daemon

‘`-d[number]`’
‘`--daemon[=number]`’
: Run in standalone mode. An optional *number* specifies the maximum number of child processes the daemon is allowed to fork. When it is omitted, it defaults to 20 processes. *Please note*, that there should be no whitespace between the ‘`-d`’ and its parameter.

‘`-i`’
‘`--inetd`’ Run in inetd mode.

‘`-p number`’
‘`--port number`’
: Listen on given port *number*. This option is meaningful only in standalone mode. It defaults to port 143.

‘`-t number`’
‘`--timeout number`’
: Set idle timeout to given *number* of seconds. The daemon breaks the connection if it receives no commands from the client within that number of seconds.

2.1.6 Auth

These options control the authorization and authentication module lists. For a description of authentication concepts, refer to See Section 2.2 [authentication], page 7.

‘`--authorization modlist`’
: This option allows to set up a list of modules to be used for authorization. *modlist* is a colon-separated list of modules. Valid modules are:

  system — User credentials are retrieved from the system user database (‘`/etc/password`’).

  sql — User credentials are retrieved from the sql database. The set of ‘`--sql-`’ options (see below) is used to configure access to the database.

  virtdomain — User credentials are retrieved from a “virtual domain” user database.

‘`--authentication modlist`’
: This option allows to set up a list of modules to be used for authentication. *modlist* is a colon-separated list of modules. Valid modules are:

  generic — The generic authentication type. User password is hashed and compared against the hash value returned in authorization stage.

  system — The hashed value of the user password is retrieved from ‘`/etc/shadow`’ file on systems that support it.

  sql — The hashed value of the user password is retrieved from the sql database using query supplied by ‘`--sql-getpass`’ option (see below).

pam
: The user is authenticated via pluggable authentication module (pam). The pam service name to be used is configured via '`--pam-service`' option (see below).

'`--pam-service` *name*'
: When compiled with pam support, this option specifies the name of pam service to be used when authenticating.

The following options exist in this group if the package was configured with '`--enable-sql`' option. They take effect only if the '`sql`' module is used in authentication and/or authorization.

'`--sql-interface` *iface*'
: Specify SQL interface to use. *Iface* is either '`mysql`' or '`postgres`'. This allows to select SQL subsystem on runtime if `mailutils` was compiled with support for several SQL drivers.

  If this option is omitted, `mailutils` will use the first available SQL driver.

'`--sql-host` *name*'
: Name or IP of MySQL server to connect to.

'`--sql-user` *name*'
: sql user name

'`--sql-passwd` *string*'
: sql connection password

'`--sql-db` *string*'
: Name of the database to connect to.

'`--sql-port` *number*'
: Port to use

'`--sql-getpwnam` *query*'
: sql query to retrieve a passwd entry based on username

'`--sql-getpwuid` *query*'
: sql query to retrieve a passwd entry based on user ID

'`--sql-getpass` *query*'
: sql query to retrieve a password from the database

2.1.7 Encryption

These options control TLS/SSL encryption in `imap4d` and `pop3d` daemons.

'`--ssl-cert` *file*'
: This option specifies the file name of the server side SSL certificate (accepts PEM format).

'`--ssl-key` *file*'
: This option specifies the file name of the server side private SSL key (accepts PEM format). The key must be protected with 0600 file permissions (u=rw,g=,o=), otherwise `imap4d` or `pop3d` daemons will refuse to support TLS/SSL encryption.

‘`--ssl-cafile` *file*’
: This option specifies a file containing the list of trusted CAs (PEM list) in order to verify client's certificates. This option is not required.

2.1.8 Logging

‘`--log-facility` *facility*’
: Output logs to the specified `syslog` facility. The following facility names are recognized: ‘`user`’, ‘`daemon`’, ‘`mail`’, ‘`auth`’ and ‘`local0`’ through ‘`local7`’. These names are case-insensitive.

2.1.9 Sieve Specific Options

The following options comprise this group:

‘`-I` *dir*’
‘`--includedir=`*dir*’
: Append directory *dir* to the list of directories searched for include files.

‘`-L` *dir*’
‘`--libdir=`*dir*’
: Append directory *dir* to the list of directories searched for library files.

2.1.10 A Sample Configuration File

The following configuration file specifies that all Mailutils programs should use ‘`/var/spool/mail`’ as a local mailspool directory. Programs performing authentication will use pam service ‘`mailutils`’. All programs, except `imap4d` will issue log messages via ‘`mail`’ facility, `imap4d` will use facility ‘`local1`’.

```
:mailbox --mail-spool /var/spool/mail
:auth --authentication pam --pam-service mailutils
:logging --log-facility mail
imap4d --daemon=20 --timeout=1800 --log-facility local1
```

2.2 Authorization and Authentication Principles

Some mail utilities provide access to their services only after verifying that the user is actually the person he is claiming to be. Such programs are, for example, `pop3d` and `imap4d`. The process of the verification is broken down into two stages: *authorization* and *authentication*. In *authorization* stage the program retrieves the information about a particular user. In *authentication* stage, this information is compared against the user-supplied credentials. Only if both stages succeed is the user allowed to use the service.

A set of *modules* is involved in performing each stage. For example, the authorization stage can retrieve the user description from various sources: system database, sql database, virtual domain table, etc. Each module is responsible for retrieving the description from a particular source of information. The modules are arranged in a *module list*. The modules from the list are invoked in turn, until either a one of them succeeds or the list is exhausted. In latter case the authorization fails. Otherwise the data returned by the succeeded module are used in authentication.

Similarly, authentication may be performed in several ways. The authentication modules are also grouped in a list. Each module is tried in turn until either a module succeeds, in which case the authentication succeeds, or the end of the list is reached.

We represent the module lists as column-separated lists of module names. For example, the authorization list

```
system:sql:virtdomains
```

means that first the system user database (‘`/etc/password`’) is searched for a description of a user in question. If the search fails, the sql database is searched. Finally, if it also fails, the search is performed in the virtual domain database.

Note, that some authentication and/or authorization modules may be disabled when configuring the package before compilation. The names of the disabled modules are nevertheless available for use in runtime configuration options, but they represent a “fail-only” functionality, e.g. if the package was compiled without sql support then the module ‘`sql`’ in the above example will always fail, thus passing the execution on to the next module.

The modules available for use in authorization list are:

system — User credentials are retrieved from the system user database (‘`/etc/password`’).

sql — User credentials are retrieved from the sql database. The set of ‘`--sql-`’ options (see Section 2.1.6 [auth], page 5) is used to configure access to the database.

virtdomain — User credentials are retrieved from a “virtual domain” user database.

The modules available for use in authentication list are:

generic — The generic authentication type. User password is hashed and compared against the hash value returned in authorization stage.

system — The hashed value of the user password is retrieved from ‘`/etc/shadow`’ file on systems that support it.

sql — The hashed value of the user password is retrieved from the sql database using query supplied by ‘`--sql-getpass`’ option (see Section 2.1.6 [auth], page 5).

pam — The user is authenticated via pluggable authentication module (pam). The pam service name to be used is configured via ‘`--pam-service`’ option (see Section 2.1.6 [auth], page 5)

Unless overridden by ‘`--authentication`’ command line option, the list of authentication modules is:

```
generic:system:pam:sql
```

Unless overridden by ‘`--authorization`’ command line option, the list of authorization modules is:

```
system:sql:virtdomains
```

2.3 `frm` and `from` — List Headers from a Mailbox

GNU mailutils provides two commands for listing messages in a mailbox. These are `from` and `frm`.

`frm`

The `frm` command outputs a header information of the selected messages in a mailbox. By default, `frm` reads the user's system mailbox and outputs the contents of `From` and `Subject` headers for each message. If a folder is specified in the command line, the program reads that folder rather than the default mailbox.

The program uses following option groups: See Section 2.1.2 [mailbox], page 4.

The following command line options alter the behavior of the program:

'`-f` *string*'
'`--field` *string*'
: Display the header named by *string* instead of `From Subject` pair.

'`-l`'
'`--to`'
: Include the contents of `To` header to the output. The output field order is then: `To From Subject`.

'`-n`'
'`--number`'
: Prefix each line with corresponding message number.

'`-Q`'
'`--Quiet`'
: Be very quiet. Nothing is output except error messages. This is useful in shell scripts where only the return status of the program is important.

'`-q`'
'`--query`'
: Print a message only if there are unread messages in the mailbox.

'`-S`'
'`--summary`'
: Print a summary line.

'`-s` *attr*'
'`--status` *attr*'
: Only display headers from messages with the given status. *Attr* may be one of the following: '`new`', '`read`', '`unread`'. It is sufficient to specify only first letter of an *attr*. Multiple '`-s`' options are allowed.

'`-t`'
'`--align`'
: Tidy mode. In this mode `frm` tries to preserve the alignment of the output fields. It also enables the use of BIDI algorithm for displaying subject lines that contain text in right-to-left orientation (such as Arabic or Hebrew).

`from`

The `from` utility displays sender and subject of each message in a mailbox. By default, it reads the user's system mailbox. If the program is given a single argument, it is interpreted as a user name whose mailbox is to be read. Obviously, permissions are required to access that user's mailbox, so such invocations may be used only by superuser.

Option ‘`-f`’ (‘`--file`’) instructs the program to read the given mailbox.

The full list of options, supported by `from` follows:

‘`-c`’
‘`--count`’ Prints only a count of messages in the mailbox and exit.

‘`-d`’
‘`--debug`’ Prints additional debugging outut.

‘`-s` *`string`*’
‘`--sender=`*`string`*’
Prints only mail from addresses containing the supplied string. FIXME: only `From:` header is examined.

‘`-f` *`url`*’
‘`--file=`*`url`*’
Examine mailbox from the given *url*.

2.4 `mail` — Send and Receive Mail

`Mail` is an enhanced version of standard `/bin/mail` program. As well as its predecessor, it can be used either in sending mode or in reading mode. `Mail` enters sending mode when one or more email addresses were specified in this command line. In this mode the program waits until user finishes composing the message, then attempts to send it to the specified addresses and exits. See Section 2.4.3 [Composing Mail], page 14, for a detailed description of this behavior.

If the command line contained no email addresses, `mail` switches to reading mode. In this mode it allows to read and manipulate the contents of a mailbox. The URL of the mailbox to operate upon is taken from the argument of '`--file`' command line option. If it is not specified, the user's system mailbox is assumed. For more detail, see Section 2.4.4 [Reading Mail], page 16.

2.4.1 Invoking `mail`

General usage of `mail` program is:

```
mail [option...] [address...]
```

If [*address*...] part is present, `mail` switches to mail sending mode, otherwise it operates in mail reading mode.

The program uses following option groups: See Section 2.1.2 [mailbox], page 4.

`Mail` understands following command line options:

'`-e`'
'`--exist`' Return true if the mailbox contains some messages. Return false otherwise. This is useful for writing shell scripts.

'`-E` *command*'
'`--exec=`*command*'
Execute *command* before opening the mailbox. Any number of '`--exec`' options can be given. The commands will be executed after sourcing configuration files (see Section 2.4.7 [Mail Configuration Files], page 33), but before opening the mailbox.

'`--exec`'

'`-f[`*file*`]`'
'`--file[=`*file*`]`'
Operate on mailbox *file*. If this option is not specified, the default is user's system mailbox. If it is specified without argument, the default is '`$HOME/mbox`'. *Please note*, that there should be no whitespace between the short variant of the option ('`-f`'), and its parameter. Similarly, when using long option ('`--file`'), its argument must be preceded by equal sign.

'`-F`'
'`--byname`'
Save messages according to sender. Currently this option is not implemented.

'`-H`'
'`--headers`'
Print header summary to stdout and exit.

‘`-i`’
‘`--ignore`’
: Ignore interrupts.

‘`-m` *path*’
‘`--mail-spool=`*path*’
: Set path to the mailspool directory

‘`-n`’
‘`--norc`’ Do not read the system-wide mailrc file. See Section 2.4.7 [Mail Configuration Files], page 33.

‘`-N`’
‘`--nosum`’ Do not display initial header summary.

‘`-p`’
‘`--print`’
‘`-r`’
‘`--read`’ Print all mail to standard output. It is equivalent to issuing following commands after starting ‘`mail -N`’:

```
print *
quit
```

‘`-q`’
‘`--quit`’ Cause interrupts to terminate program.

‘`-s` *subj*’
‘`--subject=`*subj*’
: Send a message with a Subject of *subj*. Valid only in sending mode.

‘`-t`’
‘`--to`’ Switch to sending mode.

‘`-u` *user*’
‘`--user=`*user*’
: Operate on *user*’s mailbox. This is equivalent to:

```
mail -f/spool_path/user
```

with *spool_path* being the full path to your mailspool directory (‘`/var/spool/mail`’ or ‘`/var/mail`’ on most systems).

‘`-?`’
‘`--help`’ Display a help message.
‘`--usage`’ Display a short usage summary.

‘`-V`’
‘`--version`’
: Print program version and exit.

2.4.2 How to Specify Message Sets

Many mail commands such as print and delete can be given a *message list* to operate upon. Wherever the message list is omitted, the command operates on the current message.

The *message list* in its simplest form is one of:

. Selects current message. It is equivalent to empty message list.

* Selects all messages in the mailbox.

^ Selects first non-deleted message.

$ Selects last non-deleted message.

In its complex form, the *message list* is a comma or whitespace-separated list of *message specifiers*. A *message specifier* is one of

Message Number
: This specifier addresses the message with the given ordinal number in the mailbox.

Message range
: *Message range* is specified as two message numbers separated by a dash. It selects all messages with the number lying within that range.

Attribute specifier
: An *Attribute specifier* is a colon followed by a single letter. The *Attribute specifier* addresses all messages in the mailbox that have the given attribute. These are the valid attribute specifiers:

  ‘`:d`’ Selects all deleted messages.

  ‘`:n`’ Selects all recent messages, i.e. the messages that have not been neither read not seen so far.

  ‘`:o`’ Selects all messages that have been seen.

  ‘`:r`’ Selects all messages that have been read.

  ‘`:u`’ Selects all messages that have *not* been read.

  ‘`:t`’ Selects all tagged messages.

  ‘`:T`’ Selects all untagged messages.

Header match
: The *header match* is a string in the form:

  ```
  [header:]/string/
  ```

  It selects all messages that contain header field *header* matching given *regexp*. If the variable `regexp` is set, the *string* is assumed to be a POSIX regexp. Otherwise, a header is considered to match *string* if the latter constitutes a substring of the former (comparison is case-insensitive).

  If *header*: part is omitted, it is assumed to be ‘`Subject:`’.

Message body match
: The *message body match* is a string in the form:

  ```
  :/string/
  ```

  It selects all messages whose body matches the string. The matching rules are the same as described under “Header match”.

A *message specifier* can be followed by *message part specifier*, enclosed in a pair of brackets. A *message part specifier* controls which part of a message should be operated upon. It is meaningful only for multipart messages. A *message part specifier* is a comma or whitespace - separated list of part numbers or ranges. Each part number can in turn be *message part specifier*, thus allowing for operating upon multiply-encoded messages.

The following are the examples of valid message lists:

2.4.3 Composing Mail

You can compose the message by simply typing the contents of it, line by line. But usually this is not enough, you would need to edit your text, to quote some messages, etc. `Mail` provides these capabilities through *compose escapes*. The *compose escapes* are single-character commands, preceded by special *escape character*, which defaults to '`~`'. The combination `escape character + command` is recognized as a compose escape only if it occurs at the beginning of a line. If the escape character must appear at the beginning of a line, enter it twice. The actual escape character may be changed by setting the value of `escape` mail variable (see Section 2.4.6 [Mail Variables], page 26).

Quitting Compose Mode

There are several commands allowing you to quit the compose mode.

Typing the end-of-file character ('`C-D`') on a line alone finishes compose mode and sends the message to its destination. The '`C-D`' character looses its special meaning if `ignoreeof` mail variable is set.

If mail variable `dot` is set, typing dot ('`.`') on a line alone achieves the same effect as '`C-D`' above.

Finally, using '`~.`' escape always quits compose mode and sends out the composed message.

To abort composing of a message without sending it, type interrupt character (by default, '`C-C`') twice. This behavior is disabled when mail variable `ignore` is set. In this case, you can use '`~x`' escape to achieve the same effect.

Getting Help on Compose Escapes: ~?

The '`~?`' escape prints on screen a brief summary of the available compose escapes. *Please note*, that '`~h`' escape prompts for changing the header values, and does *not* give help.

Editing the Message: ~e and ~v

If you are not satisfied with the message as it is, you can edit it using a text editor specified either by `EDITOR` or by `VISUAL` environment variables. The '`~e`' uses the former, and '`~v`' uses the latter.

By default both escapes allow you to edit only the body of the message. However, if the `editheaders` variable is set, `mail` will load into the editor the complete text of the message with headers included, thus allowing you to change the headers as well.

Modifying the Headers: ~h, ~t, ~c, ~b, ~s

To add new addresses to the list of message recipients, use '`~t`' command, e.g.:

```
~t name1@domain.net name2
```

To add addresses to `Cc` or `Bcc`, use '`~c`' or '`~b`' escapes respectively.

To change the `Subject` header, use '`~s`' escape, e.g.:

```
~s "Re: your message"
```

Finally, to edit all headers, type '`~h`' escape. This will present you with the values of `To`, `Cc`, `Bcc`, and `Subject` headers allowing to edit them with normal text editing commands.

Enclosing Another Message: ~m and ~M

If you are sending mail from within mail command mode, you can enclose the contents of any message sent to you by using '`~m`' or '`~M`' commands. Typing '`~m`' alone will enclose the contents of the current message, typing '`~m 12`' will enclose the contents of message #12 and so on.

The '`~m`' uses retained and ignored lists when enclosing headers, the '`~M`' encloses all header fields.

In both cases, the contents of `indentprefix` mail variable is prepended to each line enclosed.

Adding a File to the Message: ~r and ~d

To append the contents of file *filename* to the message, type

```
~r filename
```

or

```
~< filename
```

The '`~d`' escape is a shorthand for

```
~r dead.letter
```

Printing And Saving the Message

The '`~p`' escape types the contents of the message entered so far, including headers, on your terminal. You can save the message to an arbitrary file using '`~w`' escape. It takes the filename as its argument.

Signing the Message: ~a and ~A

To save you the effort of typing your signature at the end of each message, you can use '`~a`' or '`~A`' escapes. If your signature occupies one line only, save it to the variable `sign` and use '`~a`' escape to insert it. Otherwise, if it is longer than one line, save it to a file, store the name of this file in the variable `Sign`, and use '`~A`' escape to insert it into the message.

Printing Another Message: ~f and ~F

Sometimes it is necessary to view the contents of another message, while composing. These two escapes allow it. Both take the message list as their argument. If they are used without argument, the contents of the current message is printed. The difference between '`~f`' and '`~F`' is that the former uses ignored and retained lists to select headers to be displayed, whereas the latter prints all headers.

Inserting Value of a Mail Variable: ~i

The '~i' escape enters the value of the named mail variable into the body of the message being composed.

Executing Other Mail Commands: ~: and ~-

You can execute a mail command from within compose mode using '~:' or '~-' escapes. For example, typing

```
~: from :t
```

will display the from lines of all tagged messages. Note, that executing mail-sending commands from within the compose mode is not allowed. An attempt to execute such a command will result in diagnostic message "Command not allowed in an escape sequence" being displayed. Also, when starting compose mode immediately from the shell (e.g. running '`mail address@domain`'), most mail commands are meaningless, since there is no mailbox to operate upon. In this case, the only commands that can reasonably be used are: `alias`, `unalias`, `alternate`, `set`, and `unset`.

Executing Shell Commands: ~! and ~|

The '~!' escape executes specified command and returns you to `mail` compose mode without altering your message. When used without arguments, it starts your login shell. The '~|' escape pipes the message composed so far through the given shell command and replaces the message with the output the command produced. If the command produced no output, `mail` assumes that something went wrong and retains the old contents of your message.

2.4.4 Reading Mail

To read messages from a given mailbox, use one of the following ways of invoking `mail`:

`mail` To read messages from your system mailbox.

`mail --file`
To read messages from your mailbox ('`$HOME/mbox`').

`mail --file=`*`path_to_mailbox`*
To read messages from the specified mailbox.

`mail --user=`*`user`*
To read messages from the system mailbox belonging to *user*.

Please note, that usual mailbox permissions won't allow you to use the last variant of invocation, unless you are a super-user. Similarly, the last but one variant is also greatly affected by the permissions the target mailbox has.

Unless you have started mail with '`--norc`' command line option, it will read the contents of the system-wide configuration file. Then it reads the contents of user configuration file, if any. For detailed description of these files, see Section 2.4.7 [Mail Configuration Files], page 33. After this initial setup, `mail` displays the first page of header lines and enters interactive mode. In interactive mode, `mail` displays its prompt ('`?`', if not set otherwise) and executes the commands the user enters.

Quitting the Program

Following commands quit the program:

'`quit`' Terminates the session. If `mail` was operating upon user's system mailbox, then all undeleted and unsaved messages that have been read and are not marked with hold flag are saved to the user's mbox file ('`$HOME/mbox`'). The messages, marked with `delete` are removed. The program exits to the Shell, unless saving the mailbox fails, in which case user can escape with the exit command.

'`exit`'
'`ex`'
'`xit`' Program exits to the Shell without modifying the mailbox it operates upon.

Typing EOF ('`C-D`') alone is equivalent to '`quit`'.

Obtaining Online Help

Following commands can be used during the session to request online help:

'`help [`*`command`*`]`'
'`hel [`*`command`*`]`'
'`? [`*`command`*`]`'
Display detailed command synopsis. If no *command* is given, help for all available commands is displayed.

'`list`'
'`*`' Print a list of available commands.

'`version`'
'`ve`' Display program version.

'`warranty`'
'`wa`' Display program warranty statement.

Moving Within a Mailbox

'`next`'
'`n`' Move to the next message.

'`previous`'
'`prev`' Move to the previous message.

Changing Mailbox/Directory

'`cd [`*`dir`*`]`'
'`chdir [`*`dir`*`]`'
'`ch [`*`dir`*`]`'
Change to the specified directory. If *dir* is omitted, `$HOME` is assumed.

'`file [`*`mailbox`*`]`'
'`fi [`*`mailbox`*`]`'
'`folder [`*`mailbox`*`]`'
'`fold [`*`mailbox`*`]`'
Read in the contents of the specified *mailbox*. The current mailbox is updated as if `quit` command has been issued. If *mailbox* is omitted, the command prints

the current mailbox name followed by the summary information regarding it, e.g.:

```
& fold
"/var/spool/mail/gray": 23 messages 22 unread
```

Controlling Header Display

To control which headers in the message should be displayed, `mail` keeps two lists: a *retained* header list and an *ignored* header list. If *retained* header list is not empty, only the header fields listed in it are displayed when printing the message. Otherwise, if *ignored* header list is not empty, only the headers *not listed* in this list are displayed. The uppercase variants of message-displaying commands can be used to print all the headers.

The following commands modify and display the contents of both lists.

'`discard [`*header-field-list*`]`'
'`di [`*header-field-list*`]`'
'`ignore [`*header-field-list*`]`'
'`ig [`*header-field-list*`]`'
: Add *header-field-list* to the ignored list. When used without arguments, this command prints the contents of ignored list.

'`retain [`*header-field-list*`]`'
'`ret [`*header-field-list*`]`'
: Add *header-field-list* to the retained list. When used without arguments, this command prints the contents of retained list.

Displaying Information

'`=`' Displays the current message number.

'`headers [`*msglist*`]`'
'`h [`*msglist*`]`'
: Lists the current pageful of headers.

'`from [`*msglist*`]`'
'`f [`*msglist*`]`'
: Lists the contents of '`From`' headers for a given set of messages.

'`z [`*arg*`]`' Presents message headers in pagefuls as described for `headers` command. When *arg* is '`.`', it is generally equivalent to `headers`. When *arg* is omitted or is '`+`', the next pageful of headers is displayed. If *arg* is '`-`', the previous pageful of headers is displayed. The latter two forms of `z` command may also take a numerical argument meaning the number of pages to skip before displaying the headers. For example:

```
& z +2
```

will skip two pages of messages before displaying the header summary.

'`size [`*msglist*`]`'
'`si [`*msglist*`]`'
: Lists the message number and message size in bytes for each message in *msglist*.

‘`folders`’ Displays the value of `folder` variable.

‘`summary`’
‘`su`’ Displays current mailbox summary. E.g.:

```
& summary
"/var/spool/mail/gray": 23 messages 22 unread
```

Displaying Messages

‘`print [`*msglist*`]`’
‘`p [`*msglist*`]`’
‘`type [`*msglist*`]`’
‘`t [`*msglist*`]`’
Prints out the messages from *msglist*. The variable `crt` determines the minimum number of lines the body of the message must contain in order to be piped through pager command specified by environment variable `PAGER`. If `crt` is set to a numeric value, this value is taken as the minimum number of lines. Otherwise, if `crt` is set without a value then the height of the terminal screen is used to compute the threshold. The number of lines on screen is controlled by `screen` variable.

‘`Print [`*msglist*`]`’
‘`P [`*msglist*`]`’
‘`Type [`*msglist*`]`’
‘`T [`*msglist*`]`’
Like print but also prints out ignored header fields.

‘`decode [`*msglist*`]`’
‘`dec [`*msglist*`]`’
Print a multipart message. The `decode` command decodes and prints out specified message parts. E.g.

```
& decode 15[2]
+---------------------------------------
| Message=15[2]
| Type=message/delivery-status
| encoding=7bit
+---------------------------------------
Content-Type: message/delivery-status
...
```

‘`top [`*msglist*`]`’
‘`to [`*msglist*`]`’
Prints the top few lines of each message in *msglist*. The number of lines printed is controlled by the variable `toplines` and defaults to five.

‘pipe [*msglist*] [*shell-command*]’
‘| [*msglist*] [*shell-command*]’
Pipe the contents of specified messages through *shell-command*. If *shell-command* is empty but the string variable `cmd` is set, the value of this variable is used as a command name.

Marking Messages

‘tag [*msglist*]’
‘ta [*msglist*]’
Tag messages. The tagged messages can be referred to in message list using ‘:t’ notation.

‘untag [*msglist*]’
‘unt [*msglist*]’
Clear tags from specified messages. To untag all messages tagged so far type

```
& untag :t
```

‘hold [*msglist*]’
‘ho [*msglist*]’
‘preserve [*msglist*]’
‘pre [*msglist*]’
Marks each message to be held in user's system mailbox. This command does not override the effect of `delete` command.

Disposing of Messages

‘delete [*msglist*]’
‘d [*msglist*]’
Mark messages as deleted. Upon exiting with `quit` command these messages will be deleted from the mailbox. Until the end of current session the deleted messages can be referred to in message lists using :d notation.

‘undelete [*msglist*]’
‘u [*msglist*]’
Clear delete mark from the specified messages.

‘dp [*msglist*]’
‘dt [*msglist*]’
Deletes the current message and prints the next message. If *msglist* is specified, deletes all messages from the list and prints the message, immediately following last deleted one.

Saving Messages

‘save [[*msglist*] *file*]’
‘s [[*msglist*] *file*]’
Takes a message list and a file name and appends each message in turn to the end of the file. The name of file and number of characters appended to it is echoed on the terminal. Each saved message is marked for deletion as if with `delete` command, unless the variable `keepsave` is set.

'`Save [msglist]`'
'`S [msglist]`'
Like `save`, but the file to append messages to is named after the sender of the first message in *msglist*. For example:

```
& from 14 15
 U  14 smith@noldor.org Fri Jun 30 18:11  14/358   The Save c
 U  15 gray@noldor.org  Fri Jun 30 18:30  8/245    Re: The Sa
& Save 14 15
"smith" 22/603
```

i.e., 22 lines (603 characters) have been appended to the file "smith". If the file does not exist, it is created.

'`write [[msglist] file]`'
'`w [[msglist] file]`'
Similar to `save`, except that only message body (without the header) is saved.

'`Write [msglist]`'
'`W [msglist]`'
Similar to `Save`, except that only message body (without the header) is saved.

'`mbox [msglist]`'
'`mb [msglist]`'
'`touch [msglist]`'
'`tou [msglist]`'
Mark list of messages to be saved in the user's mailbox ('`$HOME/mbox`') upon exiting via `quit` command. This is the default action for all read messages, unless you have variable `hold` set.

'`copy [[msglist] file]`'
'`c [[msglist] file]`'
Similar to `save`, except that saved messages are not marked for deletion.

'`Copy [msglist]`'
'`C [msglist]`'
Similar to `Save`, except that saved messages are not marked for deletion.

Editing Messages

These command allow to edit messages in a mailbox. *Please note*, that modified messages currently do not replace original ones. i.e. you have to save them explicitly using your editor's `save` command if you do not want the effects of your editing to be lost.

'`edit [msglist]`'
'`e [msglist]`'
Edits each message in *msglist* with the editor, specified in `EDITOR` environment variable.

'`visual [msglist]`'
'`v [msglist]`'
Edits each message in *msglist* with the editor, specified in `VISUAL` environment variable.

Aliasing

‘`alias [alias [address...]]`’
‘`a [alias [address...]]`’
‘`group [alias [address...]]`’
‘`g [alias [address...]]`’
With no arguments, prints out all currently-defined aliases. With one argument, prints out that alias. With more than one argument, creates a new alias or changes an old one.

‘`unalias [alias...]`’
‘`una [alias...]`’
Takes a list of names defined by alias commands and discards the remembered groups of users. The alias names no longer have any significance.

‘`alternates name...`’
‘`alt name...`’
The alternates command is useful if you have accounts on several machines. It can be used to inform mail that the listed addresses are really you. When you reply to messages, mail will not send a copy of the message to any of the addresses listed on the alternates list. If the alternates command is given with no argument, the current set of alternate names is displayed.

Replying

‘`mail [address...]`’
‘`m [address...]`’
Switches to compose mode. After composing the message, sends messages to the specified addresses.

‘`reply [msglist]`’
‘`respond [msglist]`’
‘`r [msglist]`’
For each message in *msglist*, switches to compose mode and sends the composed message to the sender and all recipients of the message.

‘`Reply [msglist]`’
‘`Respond [msglist]`’
‘`R [msglist]`’
Like `reply`, except that the composed message is sent only to originators of the specified messages.

Notice, that setting mail variable `flipr` (see Section 2.4.6 [Mail Variables], page 26) swaps the meanings of the two above commands, so that `reply` sends the message to the sender and all recipients of the message, whereas `Reply` sends it to originators only.

‘`followup [msglist]`’
‘`fo [msglist]`’
Switches to compose mode. After composing, sends the message to the originators and recipients of all messages in *msglist*.

‘`Followup [msglist]`’
‘`F [msglist]`’
: Similar to `followup`, but reply message is sent only to originators of messages in *msglist*.

To determine the sender of the message `mail` uses the list of sender fields (see [Controlling Sender Fields], page 23). The first field from this list is looked up in message headers. If it is found and contains a valid email address, this address is used as the sender address. If not, the second field is searched and so on. This process continues until a field is found in the headers, or the sender field list is exhausted, whichever happens first.

If the previous step did not determine the sender address, the address from SMTP envelope is used.

Let's illustrate this. Suppose your mailbox contains the following:

```
 U  1 block@helsingor.org  Fri Jun 30 18:30  8/245    Re: The Sa
& Print 1
From: Antonius Block <block@helsingor.org>
To: Smeden Plog <plog@helsingor.org>
Date: Tue, 27 Apr 2004 13:23:41 +0300
Reply-To: <root@helsingor.org>
Subject: News

Hi
```

Now, you issue the following commands:

```
& sender mail-followup-to reply-to from
& reply
To: <root@helsingor.org>
Subject: Re: News
```

As you see, the value of `Reply-To` field was taken as the sender address.

Now, let's try the following command sequence:

```
# Clear the sender list
& nosender
# Set new sender list
& sender From
```

Now, the `From` address will be taken:

```
& reply
To: Antonius Block <block@helsingor.org>
Subject: Re: News
```

Controlling Sender Fields

Commands `sender` and `nosender` are used to manipulate the contents of the sender field list.

If the command `sender` is used without arguments, it displays the contents of the sender field list. If arguments are given, each argument is appended to the sender field list. For example:

```
& sender
Sender address is obtained from the envelope
& sender mail-followup-to reply-to
& sender
mail-followup-to
reply-to
& sender from
& sender
mail-followup-to
reply-to
from
```

Command `nosender` is used to remove items from the sender field list:

```
& sender
mail-followup-to
reply-to
from
& nosender reply-to
& sender
mail-followup-to
from
```

When used without arguments, this command clears the list:

```
& nosender
Sender address is obtained from the envelope
```

Incorporating New Mail

The `incorporate` (`inc`) command incorporates newly arrived messages to the displayed list of messages. This is done automatically before returning to `mail` command prompt if the variable `autoinc` is set.

Shell Escapes

To run arbitrary shell command from `mail` command prompt, use `shell` (`sh`) command. If no arguments are specified, the command starts the user login shell. Otherwise, it uses its first argument as a file name to execute and all subsequent arguments are passed as positional parameters to this command. The `shell` command can also be spelled as `!`.

2.4.5 Scripting

Comments

The '`#`' character introduces an end-of-line comment. All characters until and including the end of line are ignored.

Displaying Arbitrary Text

The ‘`echo`’ (‘`ec`’) command prints its arguments to stdout.

Sourcing External Command Files

The command ‘`source` *`filename`*’ reads commands from the named file. Its minimal abbreviation is ‘`so`’.

Setting and Unsetting the Variables

The mail variables may be set using ‘`set`’ (‘`se`’) command. The command takes a list of assignments. The syntax of an assignment is

‘*`name=string`*’
: Assign a string value to the variable. If *string* contains whitespace characters it must be enclosed in a pair of double-quote characters (‘`"`’)

‘*`name=number`*’
: Assign a numeric value to the variable.

‘*`name`*’
: Assign boolean `True` value.

‘`no`*`name`*’
: Assign boolean `False` value.

Example:

```
& set askcc nocrt indentprefix="> "
```

This statement sets `askcc` to `True`, `crt` to `False`, and `indentprefix` to “`> `”.

To unset mail variables use ‘`unset`’(‘`uns`’) command. The command takes a list of variable names to unset.

Example: To undo the effect of the previous example, do:

```
& unset askcc crt indentprefix
```

Setting and Unsetting Shell Environment Variables

Shell environment may be modified using ‘`setenv`’ (‘`sete`’) command. The command takes a list of assignments. The syntax of an assignment is:

‘*`name=value`*’
: If variable *name* does not already exist in the environment, then it is added to the environment with the value *value*. If *name* does exist, then its value in the environment is changed to *value*.

‘*`name`*’
: Delete the variable *name* from the environment (“unset” it).

Conditional Statements

The conditional statement allows to execute a set of mail commands depending on the mode the `mail` program is in. The conditional statement is:

```
if cond
...
else
...
endif
```

where ‘`...`’ represents the set of commands to be executed in each branch of the statement. *cond* can be one of the following:

‘s’ True if `mail` is operating in mail sending mode.

‘r’ True if `mail` is operating in mail reading mode.

‘t’ True if stdout is a terminal device (as opposed to a regular file).

The conditional statements can be nested to arbitrary depth. The minimal abbreviations for ‘`if`’, ‘`else`’ and ‘`endif`’ commands are ‘`i`’, ‘`el`’ and ‘`en`’.

Example:

```
if t
set crt prompt="& "
else
unset prompt
endif
if s
alt gray@farlep.net gray@mirddin.farlep.net
set
```

2.4.6 How to Alter the Behavior of `mail`

Following variables control the behavior of GNU `mail`:

`appenddeadletter`

Type: Boolean.
Default: False.

If this variable is `True`, the contents of canceled letter is appended to the user's ‘`dead.letter`’ file. Otherwise it overwrites its contents.

`askbcc`

Type: Boolean.
Default: False.

When set to `True` the user will be prompted to enter `Bcc` field before composing the message.

`askcc`

Type: Boolean.
Default: True.

When set to `True` the user will be prompted to enter `Cc` field before composing the message.

`asksub`

Type: Boolean.
Default: True in interactive mode, False otherwise.

When set to `True` the user will be prompted to enter `Subject` field before composing the message.

`autoinc`

Type: Boolean.
Default: True.

Automatically incorporate newly arrived messages.

`autoprint`

Type: Boolean.
Default: False.

Causes the delete command to behave like dp - thus, after deleting a message, the next one will be typed automatically.

`bang`

Type: Boolean.
Default: False.

When set, every occurrence of `!` in arguments to `!` command is replaced with the last executed command.

`datefield`

Type: Boolean.
Default: False.

By default the date in a header summary is taken from the SMTP envelope of the message. Setting this variable tells `mail` to use the date from `Date:` header field, converted to localtime. Notice, that for messages lacking this field `mail` will fall back to using SMTP envelope.

`charset`

Type: string
Default: '`auto`'

The value of this variable controls the output character set for the header fields encoding using RFC 2047. If the variable is unset, no decoding is performed and the fields are printed as they are. If the variable is set to '`auto`', `mail` tries to deduce the name of the character set from the value of `LC_ALL` environment variable. Otherwise, its value is taken as the name of the charset.

`cmd`

Type: String.
Default: Unset.

Contains default shell command for `pipe`.

`columns`

Type: Numeric.
Default: Detected at startup by querying the terminal device. If this fails, the value of environment variable `COLUMNS` is used.

This variable contains the number of columns on terminal screen.

`crt`

Type: Boolean or Numeric
Default: True in interactive mode, False otherwise.

The variable `crt` determines the minimum number of lines the body of the message must contain in order to be piped through pager command specified by environment variable `PAGER`. If `crt` is set to a numeric value, this value is taken as the threshold. Otherwise, if `crt` is set without a value, then the height

of the terminal screen is used to compute the threshold. The number of lines on screen is controlled by `screen` variable.

`decode-fallback`

Type: String.
Default: '`none`'.

This variable controls the way to represent characters that cannot be rendered using current character set. It can have three values:

'`none`' Such characters are not printed at all. The conversion process stops at the first character that cannot be rendered.

'`copy-pass`'
The characters are displayed '`as is`'. Notice, that depending on your setup, this may screw-up your terminal settings.

'`copy-octal`'
Unprintable characters are represented by their octal codes. Printable ones are printed '`as is`'.

`dot`

Type: Boolean.
Default: False.

If `True`, causes `mail` to interpret a period alone on a line as the terminator of a message you are sending.

`emptystart`

Type: Boolean.
Default: False.

If the mailbox is empty, `mail` normally prints '`No mail for user`' and exits immediately. If this option is set, `mail` will start no matter is the mailbox empty or not.

`editheaders`

Type: Boolean.
Default: False.

When set, `mail` will include message headers in the text to be the `~e` and `~v` escapes, thus allowing you to customize the headers.

`escape`

Type: String.
Default: `~`

If defined, the first character of this option gives the character to denoting escapes.

`flipr`

Type: Boolean
Default: Unset

The variable `flipr` if set swaps the meanings of `reply` and `Reply` commands (see [Replying], page 22).

`folder`

Type: String.
Default: Unset.

The name of the directory to use for storing folders of messages. If unset, `$HOME` is assumed.

`header`

Type: Boolean.
Default: True, unless started with '`--nosum`' ('`-N`') option.

Whether to run `headers` command automatically after entering interactive mode.

`hold`

Type: Boolean.
Default: False.

When set to `True`, the read or saved messages will be stored in user's mailbox ('`$HOME/mbox`'). Otherwise, they will be held in system mailbox also. This option is in effect only when operating upon user's system mailbox.

`ignore`

Type: Boolean.
Default: False.

When set to `True`, `mail` will ignore keyboard interrupts when composing messages. Otherwise an interrupt will be taken as a signal to abort composing.

`ignoreeof`

Type: Boolean.
Default: False.

Controls whether typing EOF character terminates the letter being composed.

`indentprefix`

Type: String.
Default: `"\t"` (a tab character).

String used by the `~m` tilde escape for indenting quoted messages.

`inplacealiases`

Type: Boolean
Default: False

If set, `mail` will expand aliases in the address header field before entering send mode (see Section 2.4.3 [Composing Mail], page 14). By default, the address header fields are left intact while composing, the alias expansion takes place immediately before sending message.

`keepsave`

Type: Boolean.
Default: False.

Controls whether saved messages should be kept in system mailbox too. This variable is in effect only when operating upon a user's system mailbox.

`mailx`

Type: Boolean.
Default: False.

When set, enables *mailx compatibility mode*. This mode has the following effects:

- When composing a message `mail` will ask for `Cc` and `Bcc` addresses after composing the body. The default behavior is to ask for these values before composing the body.
- In send mode, if the composition was interrupted, `mail` will exit with zero status. By default it exits with zero status only if the message was sent successfully.

`metamail`

Type: Boolean or String.
Default: True.

This variable controls operation of `decode` command. If it is unset, `decode` will not attempt any interpretation of the content of message parts. Otherwise, if `metamail` is set to `true`, `decode` will use internal metamail support to interpret message parts. Finally, if `metamail` is assigned a string, this string is treated as command line of the external `metamail` command which will be used to display parts of a multipart message. For example:

```
# Disable MIME interpretation:
set nometamail
# Enable built-in MIME support:
set metamail
# Use external program to display MIME parts:
set metamail="metamail -m mail -p"
```

`mimenoask`

Type: String
Default: Empty

By default `mail` asks for comnfirmation before running interpreter to view a part of the multi-part message. If this variable is set, its value is treated as a comma-separated list of MIME types for which no confirmation is needed. Elements of this list may include shell-style globbing patterns, e.g. setting

```
set mimenoask=text/*,image/jpeg
```

will disable prompting before displaying any textual files, no matter what their subtype is, and before displaying files with type '`image/jpeg`'.

`metoo`

Type: Boolean.
Default: False.

Usually, when an alias is expanded that contains the sender, the sender is removed from the expansion. Setting this option causes the sender to be included in the group.

`mode`

Type: String.
Default: The name of current operation mode.

Setting this variable does not affect the operation mode of the program.

`outfolder`

Type: String.
Default: Unset.

Contains the directory in which files created by `save`, `write`, etc. commands will be stored. When unset, current directory is assumed.

`page`

Type: Boolean.
Default: False.

If set to `True`, the `pipe` command will emit a linefeed character after printing each message.

`prompt`

Type: String.
Default: `"? "`

Contains the command prompt sequence.

`quit`

Type: Boolean.
Default: False, unless started with '`--quit`' ('`-q`') option.

When set, causes keyboard interrupts to terminate the program.

`rc`

Type: Boolean.
Default: True, unless started with '`--norc`' ('`-N`') option.

When this variable is set, `mail` will read the system-wide configuration file upon startup. See Section 2.4.7 [Mail Configuration Files], page 33.

`record`

Type: String.
Default: Unset.

When set, any outgoing message will be saved to the named file.

`recursivealiases`

Type: Boolean
Default: True

When set, `mail` will expand aliases recursively.

`regex`

Type: Boolean.
Default: True.

Setting this to `True` enables use of regular expressions in '`/.../`' message specifications.

`replyprefix`

Type: String
Default: '`Re: `'

Sets the prefix that will be used when constructing the subject line of a reply message.

`replyregex`

Type: String
Default: '`^re: *`'

Sets the regular expression used to recognize subjects of reply messages. If the `Subject` header of the message matches this expression, the value of `replyprefix` will not be prepended to it before replying. The expression should be a POSIX extended regular expression. The comparison is case-insensitive.

For example, to recognize usual English, Polish, Norwegian and German reply subject styles, use:

```
set replyregex="^(re|odp|aw|ang)(\\[[0-9]+\\])?:[[:blank:]]"
```

(Notice the quoting of backslash characters).

`save`

Type: Boolean.
Default: True.

When set, the aborted messages will be stored in the user's '`dead.file`'. See also `appenddeadletter`.

`screen`

Type: Numeric.
Default: Detected at startup by querying the terminal device. If this fails, the value of environment variable `LINES` is used.

This variable contains the number of lines on terminal screen.

`sendmail`

Type: String.
Default: sendmail:/usr/lib/sendmail

Contains the URL of mail transport agent.

`Sign`

Type: String.
Default: Unset.

Contains the filename holding users signature. The contents of this file is appended to the end of a message being composed by `~A` escape.

`sign`
: Type: String.
Default: Unset.

Contains the user's signature. The contents of this variable is appended to the end of a message being composed by `~a` escape. Use `Sign` variable, if your signature occupies more than one line.

`showto`
: Type: Boolean
Default: unset

If this variable is set, `mail` will show `To:` addresses instead of `From:` for all messages that come from the user that invoked the program.

`subject`
: Type: String.
Default: Unset.

Contains default subject line. This will be used when `asksub` is off.

`toplines`
: Type: Numeric.
Default: 5

Number of lines to be displayed by `top` and `Top` commands.

`verbose`
: Type: Boolean.
Default: False.

When set, the actual delivery of messages is displayed on the user's terminal.

`xmailer`
: Type: Boolean.
Default: Set.

Controls whether the header '`X-Mailer`' should be added to outgoing messages. The default value of this header is

```
X-Mailer: mail (GNU Mailutils 1.0)
```

2.4.7 Personal and System-wide Configuration Files

Upon startup, `mail` reads the contents of the two command files: the system-wide configuration file, and the user's configuration file. Each line read from these files is processed like a usual `mail` command.

When run with '`--norc`' ('`-N`') option, `mail` does not read the contents of system-wide configuration file. The user's file, if it exists, is always processed.

The user's configuration file is located in the user's home directory and is named '`.mailrc`'. The location and name of the system-wide configuration file is determined when configuring the package via '`--with-mail-rc`' option. It defaults to '*sysconfdir*`/mail.rc`'.

2.5 `messages` — Count the Number of Messages in a Mailbox

`Messages` prints on standard output the number of messages contained in each folder specified in command line. If no folders are specified, it operates upon user's system mailbox. For each folder, the following output line is produced:

```
Number of messages in folder: number
```

where *folder* represents the folder name, *number* represents the number of messages.

The program uses following option groups: See Section 2.1.2 [mailbox], page 4.

The program accepts following command line options:

'`-q`'
'`--quiet`'
'`-s`'
'`--silent`'
: Be quiet. Display only number of messages per mailbox, without leading text.

'`-?`'
'`--help`'
: Output help message and exit.

'`--usage`'
: Output short usage summary and exit.

'`-V`'
'`--version`'
: Output program version and exit.

2.6 `movemail` — Moves Mail from the User Maildrop to the Local File

The purpose of `movemail`, as its name implies, is to move mail from one location to another. For example, the following invocation:

```
movemail /var/mail/smith INBOX
```

moves messages from file '`/var/mail/smith`' to file '`INBOX`'.

You will probably never have to run this program manually. It is intended as a replacement for `movemail` from GNU Emacs. The `movemail` program is run by Emacs `Rmail` module. See section "Rmail" in *Reading Mail with Rmail*, for detailed description of `Rmail` interface.

Mailutils version of `movemail` is completely backward-compatible with its Emacs predecessor, so it should run flawlessly with older versions of Emacs. Emacs version 21.4, which is being developed at the time of this writing, will contain improved `Rmail` interface for work with mailutils `movemail`.

2.6.1 Movemail Options

This subsection discusses `movemail` options from the point of view of an Emacs `Rmail` user.

To set various options to `movemail` from `Rmail`, use `rmail-movemail-flags` variable, or '`Rmail Movemail Flags`' section from the menu.

Some POP servers return messages in reversed order. To fix the order, use '`-p`' option or its synonym '`--reverse`'.

If the remote server supports TLS encryption, use '`--tls`' to instruct `movemail` to initiate encrypted connection.

Quite a few options control how `movemail` handles mail locking (a way of preventing simultaneous access to the source mailbox). By default, before accessing mailbox *file*, `movemail` will first see if the file named '*`file`*`.lock`' (so called *lock file*) exists. If so, it will assume that the mailbox is being used by another program and will sleep one second. If '*`file`*`.lock`' file disappears after this wait period, the program will proceed. Otherwise, it will repeat this action ten times. If after ten wait periods the lock file does not disappear, `movemail` gives up and exits.

If the lock file does not exist, `movemail` will create it, thereby indicating to other programs that the mailbox is being used, and will proceed to copying messages to the destination file. When finished, `movemail` closes the mailbox and removes the lock file.

Several options control this behavior. To change the default sleep period use '`--lock-retry-timeout`'. Its argument is the timeout value in seconds.

To change number of retries, use '`--lock-retry-count`'. For example, setting `rmail-movemail-flags` to

```
--lock-retry-timeout=2 --lock-retry-count=5
```

instructs `movemail` to make five attempts to acquire the lock file, with two-second intervals between the attempts.

You may also force `movemail` to remove the lock file if it is older than a given amount of time (a so called *stale lock file*). To do so, use the following option:

```
--lock-expire-timeout=seconds
```

The '`--lock-expire-timeout`' sets the number of seconds after which a lock file is considered stale.

There are special programs that can be used to lock and unlock mailboxes. A common example of such programs is `dotlock`. If you wish to use such *external locking program* instead of the default mailutils locking mechanism, use option '`--external-locker`'. Argument to this option specifies the full name of the external program to use.

2.6.2 Summary of Movemail Usage

```
movemail [option...] inbox destfile [remote-password]
```

The first argument, *inbox*, is the url (see Section 3.1.14 [URL], page 104) of the source mailbox. The second argument, *destfile*, traditionally means destination file, i.e. the UNIX mailbox to copy messages to. However, mailutils `movemail` extends the meaning of this parameter. You may actually specify any valid url as *destfile* parameter.[1]. Finally, optional third argument is a traditional way of specifying user passwords for remote (POP or IMAP) mailboxes.

Following is the summary of available command line options:

'`--emacs`' Output information used by Emacs rmail interface

'`-p`'
'`--preserve`'
'`--keep-messages`'
: Preserve the source mailbox

'`-r`'
'`--reverse`'
: Reverse the sorting order

'`--license`'
: Print GPL license and exit

'`--external-locker=`*`program`*'
: Use given *program* as the external locker program.

'`--lock-expire-timeout=`*`seconds`*'
: Set number of seconds after which the lock expires

'`--lock-flags=`*`flags`*'
: Set locker flags. *flags* is composed of the following letters: '`E`' – use external locker program `dotlock`, '`R`' – retry 10 times if acquiring of the lock failed (see also '`--lock-retry-count`' below), '`T`' – remove stale locks after 10 minutes (see also '`--lock-expire-timeout`', and '`P`' – write process PID to the lock file.

'`--lock-retry-count=`*`number`*'
: Set the maximum number of times to retry acquiring the lockfile

'`--lock-retry-timeout=`*`seconds`*'
: Set timeout for acquiring the lockfile

'`-m` *`url`*'
'`--mail-spool` *`URL`*'
: Use specified URL as a mailspool directory

[1] Rmail does not use this feature

‘`--tls[=bool]`’
: Enable (default) or disable TLS support

2.7 `readmsg` — Extract Messages from a Folder

The program, readmsg, extracts with the selection argument messages from a mailbox. Selection can be specify by:

1. A lone "*" means select all messages in the mailbox.
2. A list of message numbers may be specified. Values of "0" and "$" in the list both mean the last message in the mailbox. For example:

   ```
   readmsg 1 3 0
   ```

   extracts three messages from the folder: the first, the third, and the last.
3. Finally, the selection may be some text to match. This will select a mail message which exactly matches the specified text. For example,

   ```
   readmsg staff meeting
   ```

   extracts the message which contains the words "staff meeting." Note that it will not match a message containing "Staff Meeting" - the matching is case sensitive. Normally only the first message which matches the pattern will be printed.

Command line options

'`-a`'
'`--show-all`'
: If a pattern is use for selection show all messages that match pattern by default only the first one is presented.

'`-d`'
'`--debug`'
: Display mailbox debugging information.

'`-f` *`mailbox`*'
'`--folder=`*`mailbox`*'
: Specified the default mailbox.

'`-h`'
'`--header`'
: Show the entire header and ignore the weedlist.

'`-n`'
'`--no-header`'
: Do not print the message header.

'`-p`'
'`--form-feed`'
: Put form-feed (Control-L) between messages instead of newline.

'`-w` *`weedlist`*'
'`--weedlist=`*`weedlist`*'
: A whitespace or coma separated list of header names to show per message. Default is –weedlist="From Subject Date To CC Apparently-"

2.8 sieve

Sieve is a language for filtering e-mail messages at time of final delivery, described in RFC 3028. GNU Mailutils provides two implementations of this language: a stand-alone *sieve interpreter* and a *sieve translator and filter*. The following sections describe these utilities in detail.

2.8.1 A Sieve Interpreter

Sieve interpreter `sieve` allows to apply Sieve scripts to an arbitrary number of mailboxes. GNU `sieve` implements a superset of the Sieve language as described in RFC 3028. See Chapter 4 [Sieve Language], page 135, for a description of the Sieve language. See Section 4.8 [GNU Extensions], page 148, for a discussion of differences between the GNU implementation of Sieve and its standard.

Invoking sieve

The `sieve` invocation syntax is:

```
sieve [options] script
```

where *script* denotes the filename of the sieve program to parse, and *options* is one or more of the following:

'`-c`'
'`--compile-only`'
: Compile script and exit.

'`-d[flags]`'
'`--debug[=flags]`'
: Specify debug flags. The *flags* argument is a sequence of one or more of the following letters:

'`g`'	Enable main parser traces
'`T`'	Enable mailutil traces
'`P`'	Trace network protocols
'`t`'	Enable sieve trace
'`i`'	Trace the program instructions

'`-D`'
'`--dump`'
: Compile the script, dump disassembled code on standard output and exit.

'`-e address`'
'`--email address`'
: Override the user email address. This is useful for `reject` and `redirect` actions. By default, the user email address is deduced from the user name and the full name of the machine where sieve is executed.

'`-f`'
'`--mbox-url=mbox`'
: Mailbox to sieve (defaults to user's system mailbox)

'`-k`'
'`--keep-going`'
: Keep on going if execution fails on a message

‘`-n`’
‘`--no-actions`’
: Dry run: do not execute any actions, just print what would be done.

‘`-t` *ticket*’
‘`--ticket=`*ticket*’
: Ticket file for mailbox authentication

‘`-v`’
‘`--verbose`’
: Log all actions executed.

Apart from these, `sieve` understands the options from the following groups: `sieve`, `mailbox`, `mailer`, `logging`.

Logging and debugging

The default behavior of `sieve` is to remain silent about anything except errors. However, it is sometimes necessary to see which actions are executed and on which messages. This is particularly useful when debugging the sieve scripts. The ‘`--verbose`’ (‘`-v`’) option outputs log of every action executed.

Option ‘`--debug`’ allows to produce even more detailed debugging information. This option takes an argument specifying the debugging level to be enabled. The argument can consist of the following letters:

‘‘`t`’’
: This flag enables sieve tracing. It means that every test will be logged when executed.

‘‘`T`’’
: This flag enables debugging of underlying `mailutils` library.

‘‘`P`’’
: Trace network protocols: produces log of network transactions executed while running the script.

‘‘`g`’’
: Enable main parser traces. This is useful for debugging the sieve grammar.

‘‘`i`’’
: Trace the program instructions. It is the most extensive debugging level. It produces the full execution log of a sieve program, showing each instruction and states of the sieve machine. It is only useful for debugging the code generator.

Note, that there should be no whitespace between the short variant of the option (‘`-d`’), and its argument. Similarly, when using long option (‘`--debug`’), its argument must be preceded by equal sign.

If the argument to ‘`--debug`’ is omitted, it defaults to ‘`TPt`’.

Option ‘`--dump`’ produces the disassembled dump of the compiled sieve program.

By default `sieve` output all diagnostics on standard error and verbose logs on standard output. This behaviour is changed when ‘`--log-facility`’ is given in the command line (see Section 2.1.8 [logging], page 7). This option causes `sieve` to output its diagnostics to the given syslog facility.

Extending `sieve`

The basic set of sieve actions, tests and comparators may be extended using loadable extensions. Usual `require` mechanism is used for that.

When processing arguments for `require` statement, `sieve` uses the following algorithm:

1. Look up the name in a symbol table. If the name begins with '`comparator-`' it is looked up in the comparator table. If it begins with '`test-`', the test table is used instead. Otherwise the name is looked up in the action table.
2. If the name is found, the search is terminated.
3. Otherwise, transform the name. First, any '`comparator-`' or '`test-`' prefix is stripped. Then, any character other than alphanumeric characters, '`.`' and '`,`' is replaced with dash ('`-`'). The name thus obtained is used as a file name of an external loadable module.
4. Try to load the module. The module is searched in the following search paths (in the order given):
   1. Mailutils module directory. By default it is '`$prefix/lib/mailutils`'.
   2. The value of the environment variable `LTDL_LIBRARY_PATH`.
   3. Additional search directories specified with the `#searchpath` directive.
   4. System library search path: The system dependent library search path (e.g. on Linux it is set by the contents of the file '`/etc/ld.so.conf`' and the value of the environment variable `LD_LIBRARY_PATH`).

   The value of `LTDL_LIBRARY_PATH` and `LD_LIBRARY_PATH` must be a colon-separated list of absolute directories, for example, '`"/usr/lib/mypkg:/lib/foo"`'.

   In any of these directories, `sieve` first attempts to find and load the given filename. If this fails, it tries to append the following suffixes to the file name:
   1. the libtool archive extension '`.la`'
   2. the extension used for native dynamic libraries on the host platform, e.g., '`.so`', '`.sl`', etc.
5. If the module is found, `sieve` executes its initialization function (see below) and again looks up the name in the symbol table. If found, search terminates successfully.
6. If either the module is not found, or the symbol wasn't found after execution of the module initialization function, search is terminated with an error status. `sieve` then displays the following diagnostic message:

   ```
   source for the required action NAME is not available
   ```

2.8.2 A Sieve to Scheme Translator and Filter

A Sieve to Scheme Translator `sieve.scm` translates a given Sieve script into an equivalent Scheme program and optionally executes it. The program itself is written in Scheme and requires presence of Guile 1.4 on the system. For more information on Guile refer to section "Overview" in *The Guile Reference Manual*.

'`-f` *filename*'
'`--file` *filename*'
: Set input file name.

'`-o` *filename*'
'`--output` *filename*'
: Set output file name

'`-L` *dirname*'
'`--lib-dir` *dirname*'
: Set sieve library directory name

'`-d` *level*'
'`--debug` *level*'
: Set debugging level

The Scheme programs produced by `sieve.scm` can be used with `guimb` or `mail.local`.

2.9 guimb — A Mailbox Scanning and Processing Language

`Guimb` is for mailboxes what `awk` is for text files. It processes mailboxes, applying the user-supplied scheme procedures to each of them in turn and saves the resulting output in mailbox format.

The program uses following option groups: See Section 2.1.2 [mailbox], page 4.

Specifying Scheme Program to Execute

The Scheme program or expression to be executed is passed to `guimb` via the following options:

'`-s` *file*'
'`--source` *file*'
: Load Scheme source code from *file*.

'`-c` *expr*'
'`--code` *expr*'
: Execute given scheme expression.

The above switches stop further argument processing, and pass all remaining arguments as the value of `(command-line)`.

If the remaining arguments must be processed by `guimb` itself, use following options:

'`-e` *expr*'
'`--expression` *expr*'
: Execute scheme expression.

'`-f` *file*'
'`--file` *file*'
: Load Scheme source code from *file*.

You can specify both of them. In this case, the *file* is read first, then *expr* is executed. You may still pass any additional arguments to the script using '`--guile-arg`' option.

Specifying Mailboxes to Operate Upon

There are four basic ways of passing mailboxes to `guimb`.

`guimb [`*options*`] [`*mailbox...*`]`
: The resulting mailbox is not saved, unless the user-supplied scheme program saves it.

`guimb [`*options*`] --mailbox` *defmbox*
: The contents of *defmbox* is processed and is replaced with the resulting mailbox contents. Useful for applying filters to user's mailbox.

`guimb [`*options*`] --mailbox` *defmbox mailbox* `[`*mailbox...*`]`
: The contents of specified mailboxes is processed, and the resulting mailbox contents is appended to *defmbox*.

`guimb [`*options*`] --user` *username* `[`*mailbox...*`]`
: The contents of specified mailboxes is processed, and the resulting mailbox contents is appended to the user's system mailbox. This allows to use `guimb` as a mail delivery agent.

If no mailboxes are specified in the command line, `guimb` reads and processes the system mailbox of the current user.

Passing Options to Scheme

Sometimes it is necessary to pass some command line options to the scheme procedure. There are three ways of doing so.

When using '`--source`' ('`-s`') or '`--code`' ('`-c`') options, all the rest of the command line following the option's argument is passed to Scheme program verbatim. This allows for making guimb scripts executable by the shell. If your system supports '`#!`' magic at the start of scripts, add the following two lines to the beginning of your script to allow for its immediate execution:

```
#! /usr/local/bin/guimb -s
!#
```

(replace '`/usr/local/bin/`' with the actual path to the `guimb`).

Otherwise, if you use '`--file`' or '`--expression`' options, the additional arguments may be passed to the Scheme program '`-g`' ('`--guile-arg`') command line option. For example:

```
guimb --guile-arg -opt --guile-arg 24 --file progfile
```

In this example, the scheme procedure will see the following command line:

```
progfile -opt 24
```

Finally, if there are many arguments to be passed to Scheme, it is more convenient to enclose them in '`-{`' and '`-}`' escapes:

```
guimb -{ -opt 24 -} --file progfile
```

Command Line Option Summary

This is a short summary of the command line options available to `guimb`.

'`-d`'
'`--debug`' Start with debugging evaluator and backtraces.

'`-e` *expr*'
'`--expression` *expr*'
Execute given Scheme expression.

'`-m` *path*'
'`--mail-spool=`*path*'
Set path to the mailspool directory

'`-f` *progfile*'
'`--file` *progfile*'
Read Scheme program from *progfile*.

'`-g` *arg*'
'`--guile-command` *arg*'
Append *arg* to the command line passed to Scheme program.

'`-{ ... -}`'
Pass all command line options enclosed between '`-{`' and '`-}`' to Scheme program.

‘`-m`’
‘`--mailbox` *`mbox`*’
: Set default mailbox name.

‘`-u`’
‘`--user` *`name`*’
: Act as local MDA for user *name*.

‘`-h`’
‘`--help`’ Display help message.

‘`-v`’
‘`--version`’
: Display program version.

2.10 `mail.local` — Deliver Mail to the Local Mailbox

`mail.local` reads the standard input up to an end-of-file and appends the received data to the local mailboxes.

2.10.1 Invoking `mail.local`

General usage of `mail.local` program is:

```
mail.local [option...] recipient [recipient ...]
```

If recipient part is present is a FQDN, `mail.local` will attempt to deliver to a virtual host.

The program uses following option groups: See Section 2.1.2 [mailbox], page 4, See Section 2.1.6 [auth], page 5, See Section 2.1.8 [logging], page 7, See Section 2.8 [sieve], page 39.

‘`-f` *addr*’
‘`--from` *addr*’
: Specify the sender's name. This option forces `mail.local` to add ‘`From `’ envelope to the beginning of the message. If it is not specified, `mail.local` first looks into the first line from the standard input. If it starts with ‘`From `’, it is assumed to contain a valid envelope. If it does not, `mail.local` creates the envelope by using current user name and date.

‘`-h`’
‘`--help`’ Display usage summary and exit.

‘`-L`’
‘`--license`’
: Display GNU General Public License and exit.

‘`-m` *path*’
‘`--mail-spool` *path*’
: Specify path to mailspool directory.

‘`-q`’
‘`--quota-db` *file*’
: Specify path to DBM mailbox quota database (see Section 2.10.3 [Mailbox Quotas], page 48).

‘`--quota-query`’
: Specify SQL query that should be used to obtain user mailbox quotas from the SQL database (see Section 2.10.3 [Mailbox Quotas], page 48).

‘`-s` *pattern*’
‘`--source` *pattern*’
: Set name pattern for user-defined mail filters written in Scheme (see Section 2.10.5 [Scheme Filters], page 50). The metacharacters ‘`%u`’ and ‘`%h`’ in the pattern are expanded to the current recipient user name and home directory correspondingly.

  This option is available only if the package has been configured to use Guile extension language.

'`-S pattern`'
'`--sieve pattern`'
: Set name pattern for user-defined mail filters written is Sieve (see Section 2.10.4 [Sieve Filters], page 50). The metacharacters '`%u`' and '`%h`' in the pattern are expanded to the current recipient user name and home directory correspondingly.

'`-t number`'
'`--timeout number`'
: Wait *number* seconds for acquiring the lockfile. If it doesn't become available after that amount of time, return failure. The timeout defaults to 5 minutes.

'`-x flags`'
'`--debug flags`'
: Enable debugging. The debugging information will be output using syslog. The *flags* is a string consisting of the following flags: Debug flags are:

  '`g`' Start with guile debugging evaluator and backtraces. This is convenient for debugging user-defined filters (see Section 2.10.5 [Scheme Filters], page 50).

  '`T`' Enable libmailutil traces (`MU_DEBUG_TRACE`).

  '`P`' Enable network protocol traces (`MU_DEBUG_PROT`)

  '`t`' Enable sieve trace (`MU_SIEVE_DEBUG_TRACE`)

  '`l`' Enable sieve action logs

  The digits in the range '`0`' – '`9`' used in *flags* set `mail.local` debugging level.

'`-v`'
'`--version`'
: Display program version and exit.

'`--ex-multiple-delivery-success`'
: Don't return errors when delivering to multiple recipients.

'`--ex-quota-tempfail`'
: Return temporary failure if disk or mailbox quota is exceeded. By default, 'service unavailable' is returned if the message exceeds the mailbox quota.

2.10.2 Using `mail.local` with Various MTAs

This section explains how to invoke `mail.local` from configuration files of various Mail Transport Agents.

All examples in this section suppose that `mail.local` must receive following command line switches:

```
-s %h/.filter.scm -q /etc/mail/userquota
```

Using `mail.local` with Sendmail

The `mail.local` must be invoked from the local mailer definition in the '`sendmail.cf`' file. It must have the following flags set '`lswS`', meaning the mailer is local, the quote

characters should be stripped off the address before invoking the mailer, the user must have a valid account on this machine and the userid should not be reset before calling the mailer. Additionally, '`fn`' flags may be specified to allow `mail.local` to generate usual '`From `' envelope instead of the one supplied by sendmail.

If you wish to use mail.local with SQL authentication, you may wish to remove the '`w`' flag, since in that case the user is not required to have a valid account on the machine that runs `sendmail`.

Here is an example of mailer definition in '`sendmail.cf`'

```
Mlocal, P=/usr/local/libexec/mail.local,
        F=lsDFMAw5:/|@qSPfhn9,
        S=EnvFromL/HdrFromL, R=EnvToL/HdrToL,
        T=DNS/RFC822/X-Unix,
        A=mail $u
```

To define local mailer in '`mc`' source file, it will suffice to set:

```
define(‘LOCAL_MAILER_PATH’, ‘/usr/local/libexec/mail.local’)
define(‘LOCAL_MAILER_ARGS’, ‘mail $u’)
```

Using `mail.local` with Exim

Using `mail.local` with Exim is quite straightforward. The following example illustrates the definition of appropriate transport and director in '`exim.conf`':

```
# transport
mail_local_pipe:
  driver = pipe
  command = /usr/local/libexec/mail.local $local_part
  return_path_add
  delivery_date_add
  envelope_to_add

# director
mail_local:
  driver = localuser
  transport = mail_local_pipe
```

2.10.3 Setting up Mailbox Quotas

Sometimes it is necessary to limit the maximum size of a user's mailbox. Such maximum size is called *mailbox quota* for this user.

When delivering a message, `mail.local` first cheks if the mailbox quota is specified for the recipient. If so, `mail.local` computes the difference between the quota value and the actual size of the recipient's mailbox. This difference represents the maximum size of the message the receipient's mailbox is able to accomodate. Let's call it *msize*. Depending on its value, `mail.local` takes decision on whether to deliver the message. There are three possible cases:

1. *msize* equals zero. This means that the mailbox size has reached its limit). In this case the message is not delivered and the sender receives following notification message:

   user`: mailbox quota exceeded for this recipient`

2. *msize* is less than the size of the message `mail.local` is about to deliver. In this case the message is not delivered and the sender receives following notification message:

   user`: message would exceed maximum mailbox size for this recipient`

3. *msize* is greater than or equal to the size of the message. In this case `mail.local` does deliver the message.

Version 1.0 of GNU mailutils is able to retrieve mailbox quotas from a DBM or SQL database.

2.10.3.1 Keeping Quotas in DBM Database

To use the DBM quota database, your copy of `mailutils` must be compiled with DBM support (one of '`--with-gdbm`', '`--with-db2`', '`--with-ndbm`', '`--with-dbm`' options to `configure`). Examine the of `mail.local --show-config-options` if not sure.

The quota database should have the following structure:

Key — Key represents the user name. Special key '`DEFAULT`' means default quota value, i.e. the one to be used if the user is not explicitely listed in the database.

Value — The mailbox quota for this user. If it is a number, it represents the maximum mailbox size in bytes. A number may optionally be followed by '`kb`' or '`mb`', meaning kilobytes and megabytes, respectively.

A special value '`NONE`' means no mailbox size limitation for this user.

Here is an example of a valid quota database

```
# Default quota value:
DEFAULT         5mb

# Following users have unlimited mailbox size
root            NONE
smith           NONE

# Rest of users
plog            26214400
karin           10mB
```

To use the DBM database, specify its full name using '`-q`' or '`--quota-db`' in the invocation of `mail.local`. For example, in `sendmail .mc` file:

```
define(‘LOCAL_MAILER_PATH’, ‘/usr/local/libexec/mail.local’)
define(‘LOCAL_MAILER_ARGS’, ‘mail -q /etc/mail/quota.db $u’)
```

2.10.3.2 Keeping Quotas in SQL Database

Option '`--quota-query`' allows to specify a special query to retrieve the quota from the database. Currently (as of mailutils version 1.0) it is assumed that this table can be accessed using the same credentials as SQL authentication tables (See Section 2.1.5 [daemon], page 5, for the detailed discussion of '`--sql-`' options).

For example, suppose you have the following quota table:

```
create table mailbox_quota (
  user_name varchar(32) binary not null,
  quota int,
  unique (user_name)
);
```

To retrieve the quota for user `%u` you may then use the following query:

```
SELECT quota
FROM mailbox_quota
WHERE user_name='%u'
```

There is no special provisions for specifying group quotas, similar to '`DEFAULT`' in DBM databases. This is because group quotas can easily be implemented using SQL language. `Mail.local` always uses the first tuple from the set returned by mailbox quota query. So, you may add a special entry to the `mailbox_quota` table that would keep the group quota. For the following discussion, it is important that the `user_name` column for this entry be lexicographically less than any other user name in the table. Let's suppose the group quota name is '`00DEFAULT`'. Then the following query:

```
SELECT quota
FROM mailbox_quota
WHERE user_name IN ('%u','00DEFAULT')
ORDER BY user_name DESC
```

will return two tuples if user `%u` is found in `mailbox_quota`. Due to `ORDER` statement, the first tuple will contain the quota for the user, which will be used by `mail.local`. On the other hand, if user name `%u` is not present in the table, the above query will return a single tuple containing the group quota.

To summarize this, here is a working '`mailutils.rc`' entry for `mail.local`:

```
:mail.local \
      --sql-db MAILAUTH \
      --sql-host some.host.net \
      --sql-user mail.local \
      --sql-passwd guessme \
      --quota-query "SELECT quota \
                     FROM mailbox_quota \
                     WHERE user_name IN ('%u','00DEFAULT') \
                     ORDER BY user_name DESC"
```

2.10.4 Implementing User-defined Sieve Mail Filters

2.10.5 Implementing User-defined Scheme Mail Filters

2.11 `mail.remote` — Pseudo-Sendmail Interface for Mail Delivery

[FIXME]

2.12 mimeview

For each file given in its command line, `mimeview` attempts to autodetect its type and invoke an appropriate file viewer.

To detect the file type, `mimeview` uses '`mime.types`' file. This file is a part of Common UNIX Printing System, see `man mime.types` for the description of its syntax. [FIXME: provide an xref to CUPS]. By default `mimeview` searches for '`mime.types`' in '`$prefix/etc/cups/`'[1], however its exact location can be specified at runtime as well (see '`--mimetypes`' below).

Once file MIME type is successfully determined, `mimeview` consults '`mailcap`' files in order to determine how to display the file. It does so essentially in the same manner as `metamail` utility, i.e., it scans all files specified in `METAMAIL` environment variable until it finds an entry describing the desired file format or until the list of files is exhausted. If `METAMAIL` variable is not set, `mimeview` uses the following default path instead:

```
$HOME/.mailcap:/usr/local/etc/mailcap:\
 /usr/etc/mailcap:/etc/mailcap:\
 /etc/mail/mailcap:/usr/public/lib/mailcap
```

The following table summarizes options specific for `mimeview`:

'`-a[`*type-list*`]`'
'`--no-ask[=`*type-list*`]`'
: By default `mimeview` asks for comnfirmation before running interpreter to view a message. If this option is used without argument, it disables the default behavior for all message types. Otherwise, if argument *type-list* is given, it specifies a comma-separated list of MIME types for which no questions should be asked. Elements of this list may include shell-style globbing patterns, e.g. setting

  ```
  --no-ask='text/*,image/jpeg'
  ```

  will disable prompting before displaying any textual files, no matter what their subtype is, and before displaying files with type '`image/jpeg`'.

  Notice, that when the long form is used, its argument must be separated from the option by a single equal sign, as shown in the example above. When the short form ('`-a`') is used, its argument must follow the option immediately, without any intervening whitespace, e.g. '`-a'text/*'`').

'`-d[`*flags*`]`'
'`--debug[=`*flags*`]`'
: Enables debugging output. *Flags* is a sequence of characters specifying the desired debugging level. Following characters are meaningful in *flags*:

  g — Enables debugging of '`mime.types`' parser

  l — Enables debugging of '`mime.types`' lexical analyzer (warning: produces *very* copious output)

[1] The exact location is determined at configuration time by setting environment variable *DEFAULT_CUPS_CONFDIR*. On most sites running

```
./configure DEFAULT_CUPS_CONFDIR=/etc/cups
```

should be recommended.

1 Prints basic information about actions to be executed and reports about exit status of executed commands.

2 Additionally displays each file name along with its MIME type

3 Additionally traces the process of looking up the matching entry in `mailcap` files.

digits from 4 to 9
The same as 3, currently.

If *flags* are not given, the default '`9`' is assumed.

'`--metamail[=file]`'
Run `metamail` to display files, instead of using the internal mechanisms. If *file* is specified, it is taken as `metamail` command line.

'`-h`'
'`--no-interactive`'
'`--print`' This options tells `mimeview` that it should run in non-interactive mode. In this mode prompting is disabled, and the normal mailcap `command` field is not executed. Instead `mimeview` will execute the command specified in the `print` field. If there is nothing in the print field, the mailcap entry is ignored and the search continues for a matching mailcap entry that does have a `print` field.

Notice, that unlike in `metamail -h`, this option does not force `mimeview` to send the output to the printer daemon.

When used with '`--metamail`' option, this option passes '`-h`' flag to the invocation of `metamail`.

By default `mimeview` behaves as if given '`--no-interactive`' option whenever its standard input is not a tty device.

'`-n`'
'`--dry-run`'
Do not do anything, just print what whould be done. Implies '`--debug=1`', unless the debugging level is set up explicitely.

'`-t file`'
'`--mimetypes file`'
Use *file* as '`mime.types`' file. If *file* is a directory, use '*file*`/mime.types`'

2.13 POP3 Daemon

The `pop3d` daemon implements the Post Office Protocol Version 3 server.

`pop3d` has two operation modes:

Inetd
: The server is started from '`/etc/inetd.conf`' file:

```
pop3  stream tcp nowait  root  /usr/local/sbin/pop3d pop3d
```

This is the default operation mode.

Standalone
: The server runs as daemon, forking a child for each new connection. This mode is triggered by '`-d`' command line switch.

The program uses following option groups: See Section 2.1.2 [mailbox], page 4, See Section 2.1.5 [daemon], page 5, See Section 2.1.8 [logging], page 7, See Section 2.1.6 [auth], page 5.

Command line options

'`-d[`*number*`]`'
'`--daemon[=`*number*`]`'
: Run in standalone mode. An optional *number* specifies the maximum number of child processes the daemon is allowed to fork. When it is omitted, it defaults to 10 processes. *Please note*, that there should be no whitespace between the '`-d`' and its parameter.

'`-h`'
'`--help`' Display short help message and exit.

'`-i`'
'`--inetd`' Run in inetd mode.

'`-m` *path*'
'`--mail-spool=`*path*'
: Set path to the mailspool directory

'`-p` *number*'
'`--port` *number*'
: Listen on given port *number*. This option is meaningful only in standalone mode. It defaults to port 110.

'`-t` *number*'
'`--timeout` *number*'
: Set idle timeout to given *number* of seconds. Default is 600 seconds (10 minutes). The daemon breaks the connection if it receives no commands from the client within that number of seconds.

'`-v`'
'`--version`'
: Display program version and exit.

'`--undelete`'
: Remove all deletion marks from the messages after opening the mailbox.

‘`--login-delay=`*`seconds`*’
: Sets the minimum allowed delay between closing a pop3d session and opening it again with the same user name.

‘`--stat-file=`*`filename`*’
: Sets the name of the login timestamp database, used with ‘`--login-delay`’. By default, these data are kept in ‘`/var/run/pop3-login`’. Be sure to specify the file name *without* DBM-specific suffix.

2.14 IMAP4 Daemon

GNU `imap4d` is a daemon implementing IMAP4 rev1 protocol for accessing and handling electronic mail messages on a server. It can be run either as a standalone program or from '`inetd.conf`' file.

2.14.1 Namespace

GNU `imap4d` supports a notion of *namespaces* defined in RFC 2342. A namespace is a set of directories upon which the user has certain permissions. It should be understood that these permissions apply only if the underlying filesystem allows them.

The three namespaces supported by `imap4d` are:

Personal Namespace
: A namespace that is within the personal scope of the authenticated user on a particular connection. The user has all permissions on this namespace.

Other Users' Namespace
: A namespace that consists of mailboxes from the "Personal Namespaces" of other users. The user can read and list mailboxes from this namespace. However, he is not allowed to use '`%`' and '`*`' wildcards with `LIST` command, that is he can access a mailbox only if he knows exactly its location.

Shared Namespace
: A namespace that consists of mailboxes that are intended to be shared amongst users and do not exist within a user's Personal Namespace. The user has all permissions on this namespace.

By default, `imap4d` starts with the following namespaces:

Personal Namespace
: The home directory of the user, if exists.

Other Users' Namespace
: Empty

Shared Namespace
: Empty

Note, that this means that by default, a user won't be able to see or otherwise access mailboxes residing in the directories other than his own home.

To change these defaults, use '`--shared-namespace`' and '`--other-namespace`' options.

2.14.2 Starting `imap4d`

`imap4d` may run either in *standalone* or in *inetd* operation modes. When run in "standalone" mode, the server disconnects from the terminal and runs as a daemon, forking a child for each new connection.

The "inetd" mode allows to start the server from '`/etc/inetd.conf`' file. This is the default operation mode.

```
imap4  stream tcp nowait  root  /usr/local/sbin/imap4d imap4d
```

The program uses following option groups: See Section 2.1.2 [mailbox], page 4, See Section 2.1.5 [daemon], page 5, See Section 2.1.8 [logging], page 7, See Section 2.1.6 [auth], page 5.

Command Line Options

‘`--create-home-dir[=`*mode*`]`’
: If a user logs in and his home directory does not exist, create it. Optional *mode* is an octal number specifying the permissions to be set on the created directory. It is not modified by the current `umask` value. The default value for *mode* is ‘`700`’ (‘`drwx------`’ in `ls` terms).

‘`-d[`*number*`]`’
‘`--daemon[=`*number*`]`’
: Run in standalone mode. An optional *number* specifies the maximum number of child processes the daemon is allowed to fork. When it is omitted, it defaults to 20 processes. *Please note*, that there should be no whitespace between the ‘`-d`’ and its parameter.

‘`-h`’
‘`--help`’ Display short help message and exit.

‘`-i`’
‘`--inetd`’ Run in inetd mode.

‘`-m` *path*’
‘`--mail-spool=`*path*’
: Set path to the mailspool directory

‘`-O` *pathlist*’
‘`--other-namespace=`*pathlist*’
: Set the list of directories forming the “Other User’s” namespace. *pathlist* is a list of directory names separated by colons.

‘`-p` *number*’
‘`--port` *number*’
: Listen on given port *number*. This option is meaningful only in standalone mode. It defaults to port 143.

‘`-S` *pathlist*’
‘`--shared-namespace=`*pathlist*’
: Set the list of directories, forming the “Shared” namespace. *pathlist* is a list of directory names separated by colons.

‘`-t` *number*’
‘`--timeout` *number*’
: Set idle timeout to given *number* of seconds. Default is 1800 seconds (30 minutes). The daemon breaks the connection if it receives no commands from the client within that number of seconds.

‘`-v`’
‘`--version`’
: Display program version and exit.

2.15 Comsat Daemon

Comsatd is the server which receives reports of incoming mail and notifies users, wishing to get this service. It can be started either from '`inetd.conf`' or as a standalone daemon.

2.15.1 Starting `comsatd`

`Comsatd` uses following option groups: See Section 2.1.2 [mailbox], page 4, See Section 2.1.5 [daemon], page 5, See Section 2.1.8 [logging], page 7.

'`-c` *file*'
'`--config` *file*'
: Read configuration from given *file*. For more information about comsatd configuration files, see Section 2.15.2 [Configuring comsatd], page 58.

'`-d`'
'`--daemon`'
: Run as a standalone daemon.

'`-i`'
'`--inetd`'
: The server is started from '`/etc/inetd.conf`' file:

```
comsat dgram  udp wait  root  /usr/sbin/comsatd \
comsatd -c /etc/comsat.conf
```

This is the default operation mode.

'`-m` *path*'
'`--mail-spool=`*path*'
: Set path to the mailspool directory

'`-p` *number*'
'`--port` *number*'
: Specify the port number to listen on. Default is 512.

'`-v`'
'`--version`'
: Output version and exit successfully.

'`-h`'
'`--help`'
: Display short help message and exit.

2.15.2 Configuring `comsatd`

The configuration parameters for `comsatd` are kept in a single configuration file. The file uses line-oriented format: each line contains a single statement. Comments are introduced with the '`#`' sign and empty lines are ignored. You can specify the configuration file to use by using '`-c`' or '`--config`' command line switch.

The configuration file statements can logically be subdivided into *General Settings*, *Security Settings* and *Access Control Lists*. The following sections address each of these statement group in detail.

General Settings

These statements control the general behavior of the comsat daemon:

max-lines *number*
: Set maximum number of message body lines to be output.

allow-biffrc (yes | no)
: Enable or disable processing of user's '`.biffrc`' file. By default, it is enabled.

Security Settings

These statements control the way `comsatd` fights possible flooding attacks.

max-requests *number*
: Set maximum number of incoming requests per '`request-control-interval`'.

request-control-interval *number*
: Set the request control interval (seconds).

overflow-delay-time *number*
: Set the initial amount of time to sleep, after the first overflow occurs.

overflow-control-interval *number*
: Set the overflow control interval. If two consecutive overflows happen within *number* seconds, the overflow-delay-time is doubled.

Access Control Lists

Access control lists determine from which addresses `comsatd` will receive mail notification messages.

The access control lists are introduced in configuration file using keyword '`acl`'. General format for an ACL rule is

```
acl action netlist
```

Here, *action* specifies the action to be taken when a request arrives from one of the networks, listed in *netlist*. There are two possible actions: '`allow`' and '`deny`'.

The *netlist* is a whitespace-separated list of network numbers. Each network number may be specified in one of the following forms:

netnum
: Means a single host with IP address *netnum*.

netnum/netmask
netnum/masklen
'`any`'
: Denotes any IP address. It is equivalent to '`0.0.0.0/0`'.

Upon receiving a notification message, `comsatd` compares its source address against each ACL rule in the order of their appearance in the configuration file. The first rule that matches the packet determines whether the message will be processed or rejected. If no matching rule was found, the default rule applies. Currently, default rule is

```
acl allow any
```

If you don't need such behavior, specify the default rule explicitly. For example, the common use would be:

```
acl allow 127.0.0.1
acl deny any
```

which makes `comsatd` receive the notification messages from localhost only.

A per-user Configuration File

By default, when a notification arrives, `comsatd` prints subject, from headers and the first five lines from the new message to the user's tty. The user is allowed to change this behavior by using his own configuration file. This file should be located in the user's home directory and should be named '`.biffrc`'. It must be owned by the user and have its permissions bits set to 0600. (*Please note*, that the use of per-user configuration files may be disabled, by specifying '`allow-biffrc no`' in the main configuration file, see see Section 2.15.2 [Configuring comsatd], page 58).

The '`.biffrc`' file consists of a series of statements. Each statement occupies one line and defines an action to be taken upon arrival of a new mail. Very long lines may be split using '\' as the last character on the line. As usual, comments may be introduced with '`#`' character.

The actions specified in '`.biffrc`' file are executed in turn. The following actions are defined:

beep
: Produce an audible signal.

echo *string*
: Output *string* to user's terminal device.

exec *prog arglist*
: Execute program *prog* with arguments from *arglist*. *prog* must be specified with absolute pathname. It may not be a setuid or setgid program.

In the description above, *string* denotes any sequence of characters. This sequence must be enclosed in a pair of double-quotes, if it contains whitespace characters. The '\' character inside a string starts a C escape sequence. Following meta-characters may be used in strings:

$u
: Expands to username

$h
: Expands to hostname

$H{name}
: Expands to value of message header '`name`'.

$B(*c*,*l*)
: Expands to message body. *c* and *l* give maximum number of characters and lines in the expansion. When omitted, they default to 400, 5.

Example I

Dump to the user's terminal the contents of '`From`' and '`Subject`' headers followed by at most 5 lines of message body.

```
echo "Mail to \a$u@$h\a\n---\n\
From: $H{from}\n\
Subject: $H{Subject}\n\
---\n\
$B(,5)\
---\n"
```

Example II

Produce a bell, then pop up the xmessage window on display :0.0 with the text formatted in the same manner as in the previous example.

```
beep
exec /usr/X11R6/bin/xmessage \
-display :0.0 -timeout 10 "Mail to $u@$h \n---\n\
From: $H{from}\n\
Subject: $H{Subject}\n\
---\n\
$B(,5)\
---\n"
```

2.16 MH — The MH Message Handling System

The primary aim of this implementation is to provide an interface between Mailutils and Emacs using mh-e module.

To use Mailutils MH with Emacs, add the following line to your site-start.el or .emacs file:

(load "mailutils-mh")

For the information about the current state of Mailutils MH implementation please refer to file '`mh/TODO`' in the Mailutils distribution directory.

[FIXME]

2.16.1 Major differences between Mailutils MH and other MH implementations

1. All programs use usual GNU long options. The support for MH single-dash options is provided for backward compatibility;
2. UUCP addresses are not supported;
3. Mailutils supports a set of new format specifications (see Section 2.16.1.1 [Format String Diffs], page 62);
4. Mailutils provides a set of new profile variables (see Section 2.16.1.2 [Profile Variable Diffs], page 64);
5. Several programs behave differently (see Section 2.16.1.3 [Program Diffs], page 64);

2.16.1.1 New and Differing MH Format Specifications

`string decode` (*string* ***str***) [MH Format]

Decodes the input string *str* as per RFC 2047. Useful in printing '`From:`', '`To:`' and '`Subject:`' headers.

Notice that, unlike the similar NMH function, `decode` checks the value of the global profile variable `Charset` (see [Charset variable], page 64) to determine the charset to output the result in. If this variable is not set, `decode` returns its argument without any change. If this variable is set to `auto`, `decode` tries to determine the charset name from the setting of `LC_ALL` environment variable. Otherwise, the value of `Charset` is taken to be the name of the character set.

`string package` () [MH Format]

Returns package name (string '`mailutils`').

`string package_string` () [MH Format]

Returns full package string (e.g. '`GNU Mailutils 2.1`')

`string version` () [MH Format]

Returns mailutils version.

`string unre` (*string* ***str***) [MH Format]

The function removes any leading whitespace and eventual '`Re:`' prefix from its argument. Useful for creating subjects in reply messages:

```
%<{subject}Subject: Re: %(unre{subject})\\n%>
```

`void reply_regex` (*string* **r**) [MH Format]

Sets the regular expression used to recognize reply messages. The argument *r* should be a POSIX extended regular expression. Matching is case insensitive.

For example, the following invocation

```
%(reply_regex ^\(re|aw|ang|odp\)\(\\[[0-9]+\\]\)?:[[:blank:]])
```

corresponds to English '`Re`', Polish '`Odp`', Norwegian '`Aw`' or German '`Ang`', optionally followed by a number in brackets, followed by colon and any amount of whitespace. Notice proper quoting of the regex metacharacters.

See also `Reply-Regex` (see [Reply-Regex variable], page 64) and `isreply` (see [isreply MH function], page 63) below.

`boolean isreply` ([*string* **str**]) [MH Format]

If *str* is not given, the value of '`Subject:`' header is taken.

The function returns true if its argument matches the "reply subject" regular expression. This expression is set via the global profile variable `Reply-Regex` (see [Reply-Regex variable], page 64) or via the format function `reply_regex`.

This function is useful for creating '`Subject:`' headers in reply messages. For example, consider the following construction:

```
%<{subject}%(lit)%<(isreply)%?\
(profile reply-prefix)%(concat)%|%(concat Re:)%>\
%(concat{subject})%(printhdr Subject: )\n%>
```

If the '`Subject:`' header already contained reply prefix, this construct leaves it unchanged. Otherwise it prepends to it the value of `Reply-Prefix` profile variable, or, if it is unset, the string '`Re:`'.

This expression is used in default '`replcomps`' and '`replgroupcomps`' files.

`boolean rcpt` ('`to`' | '`cc`' | '`me`' | '`all`') [MH Format]

This function returns true if the given element is present in the recipient mask (as modified by '`--cc`' or '`--nocc`' options) and false otherwise. It is used in default formats for `repl` and `comp`, e.g.:

```
%(lit)%<(rcpt to)%(formataddr{to})%>
```

Notice that this means that usual '`replcomps`' file will be ignoring '`--cc`' and '`--nocc`' options, unless it has been modified as shown above.

`string concat` () [MH Format]

Appends whitespace + arg to string register.

`string printhdr` (*string* **str**) [MH Format]

Prints the value of string register, prefixed by *str*. The output is formatted as a RFC 822 header, i.e. it is split at whitespace characters nearest to the width boundary and each subsequent segment is prefixed with horizontal tabulation.

`string in_reply_to` () [MH Format]

Generates the value for '`In-reply-to:`' header according to RFC 2822.

`string references` () [MH Format]

Generates the value for '`References:`' header according to RFC 2822.

2.16.1.2 New MH Profile Variables

`MH Variable string` *Charset* [Variable]

Controls the character set in which the components decoded via the `decode` (see [decode function], page 62) format function should be output.

`MH Variable string` *Reply-Regex* [Variable]

Keeps the regular expression used to recognize reply messages. The argument should be a POSIX extended regular expression. Matching is case insensitive.

For more information, please see See [reply_regex function], page 63.

2.16.1.3 Differences in MH Program Behavior

`burst`

The utility is able to burst both RFC 934 digest messages and MIME multipart messages. It provides two additional command line options: '`--recurse`' and '`--length`'.

The '`--recurse`' option instructs the utility to recursively expand the digest.

The '`--length`' option can be used to set the minimal encapsulation boundary length for RFC 934 digests. Default length is 1, i.e. encountering one dash immediately following a newline triggers digest decoding. It is OK for messages that follow RFC 934 specification. However, many user agents do not precisely follow it, in particular, they often do not escape lines starting with a dash by '`- `' sequence. `Mailman` is one of such agents. To cope with such digests you can set encapsulation boundary length to a higher value. For example, `bounce --length=8` has been found to be sufficient for most Mailman-generated digests.

`comp`

Understands '`--build`' option.

`fmtdump`

This command is not provided. Use '`fmtcheck`' instead.

`mhl`

If the argument to `ignores` contains more than one component name it must be enclosed in double-quotes. Dangling equal sign is an error, to set a string variable to the empty value assign it an empty string, e.g.: `overflowtext=""` (see the supplied '`mhl.format`' file).

Ineractive prompting is not yet implemented.

`mhn`

- New option New option '`--compose`' forces `mhn` editing mode. This is also the default mode. This differs from the standard `mhn`, which switches to the editing mode only if no other options were given and the input file name coincides with the value of `mhdraft` environment variable.
- Show mode ('`--show`') If an appropriate mhn-show-type[/subtype] was not found, GNU `mhn` prints the decoded message content using `moreproc` variable. Standard `mhn` in this case used to print '`don't know how to display content`' diagnostic.

The default behaviour is to pipe the content to the standard input of the mhn-show-type[/subtype] command. This is altered to using a temporary file if the command contains `%f` or `%F` escapes.

- Store mode ('`--store`') If the `Content-Disposition` header contains '`filename=`', and `mhn` is invoked with '`--auto`' switch, it transforms the file name into the absolute notation and uses it only if it lies below the current mhn-storage directory. Standard `mhn` only requires that the file name do not begin with '`/`'.

  Before saving a message part, GNU `mhn` checks if the file already exists. If so, it asks whether the user wishes to rewrite it. This behaviour is disabled when '`--quiet`' option was given.

`mhparam`

The '`--all`' mode does not display commented out entries.

`repl`

Understands '`--use`' option. Disposition shell provides `use` command.

`rmm`

1. Different behaviour if one of the messages in the list does not exist:

   Mailutils `rmm` does not delete any messages. Standard `rmm` in this case deletes all messages preceeding the non-existent one.
2. The `rmmproc` profile component is not used.

`pick`

The non-standard command line syntax '`--`*`field string`*'), where *field* is any string, is deprecated. It is recognized only if `pick` is called from within another program, so that existing application continue to work. Please use the following syntax instead:

```
pick --component field --pattern string
```

New command line option '`--cflags`' allows to control the type of regular expressions used. The option must occur right before '`--pattern`' or '`--component`' option (or one of its aliases, like '`--cc`', '`--from`', etc.)

The argument to this option is a string of type specifications:

B	Use basic regular expressions
E	Use extended regular expressions
I	Ignore case
C	Case sensitive

Default is '`EI`'.

The flags remain in effect until the next occurrence of '`--cflags`' option.

Sample usage:

```
pick --cflag BC --subject '*a string'
```

The date comparison options ('`--before`' and '`--after`' accept date specifications in a wide variety of formats, e.g.:

```
pick --after 20030301
pick --after 2003-03-01
```

```
pick --after 01-mar-2003
pick --after 2003-mar-01
pick --before '1 year ago'
etc...
```

`refile`

1. Linking messages between folders goes against the logic of Mailutils, so `refile` never makes links even if called with '`--link`' option. The latter is actually a synonym for '`--copy`', which preserves the original message.
2. The '`--preserve`' option is not implemented. It is retained for backward compatibility only.
3. Message specs and folder names may be interspersed.

`sortm`

New option '`--numfield`' specifies numeric comparison for the given field.

Any number of '`--datefield`', '`--textfield`' and '`--numfield`' options may be given, thus allowing to build sort criteria of arbitrary complexity.

The order of '`--.*field`' options sets the ordering priority. This differs from the behaviour of the standard `sortm`, which always orders datefield-major, textfield-minor.

Apart from sorting the mailfolder the following actions may be specified:

'`--list`' List the ordered messages using a format string given by '`--form`' or '`--format`' option.

'`--dry-run`'
Do not actually sort messages, rather print what would have been done. This is useful for debugging purposes.

2.17 `mailutils-config` — Get the Information about the Mailutils Build

This program is designed for developers wishing to link their programs against libmailbox. It allows to examine the particulars of the current build of Mailutils and to get the command line parameters necessary for compiling and linking an application with Mailutils libraries.

Getting Compiler Flags

When invoked with the option '`--compile`', or its short form '`-c`', `mailutils-config` prints the flags that must be given to the compiler for compiling the program using Mailutils functions. An example usage:

```
cc -omyprog.o `mailutils-config --compile` myprog.c
```

Getting Loader Flags

The '`--link`', or its short form '`-l`' prints to the standard output the loader flags necessary to link a program against Mailutils libraries.

When invoked without arguments, it produces the flags necessary to link against the basic library of Mailutils: '`libmailbox`'. Arguments may be given that alter this behavior. These are:

'`auth`' Print flags to link against '`libmuauth`', the library adding new authentication methods to '`libmailbox`'.

'`guile`' Print flags to link against '`libmu_scm`', the Guile interface library.

'`mbox`' Link against `mbox` format library.

'`mh`' Link against `mh` format library.

'`maildir`' Link against `maildir` format library.

'`imap`' Link against `imap` format library.

'`pop`' Link against `pop` format library.

'`all`' Link against all Mailutils format libraries.

The order of arguments does not matter.

For example, if you wrote a program '`myprog.c`' that uses standard UNIX mailbox format, MH format and the Guile interface, then you would link it with the following command:

```
cc -omyprog myprog.o `mailutils-config --link mbox mh guile`
```

Obtaining General Build Information

The '`--info`', or '`-i`' retrieves the options (flags) used when building Mailutils. It may be used with or without arguments.

When used without arguments, it prints the list of all build flags, e.g.:

```
$ mailutils-config --info
VERSION=0.4.1
USE_LIBPAM
HAVE_LIBLTDL
WITH_GDBM
WITH_GNUTLS
WITH_GSASL
WITH_GUILE
```

```
WITH_PTHREAD
WITH_READLINE
HAVE_MYSQL
ENABLE_VIRTUAL_DOMAINS
ENABLE_IMAP
ENABLE_POP
ENABLE_MH
ENABLE_MAILDIR
ENABLE_SMTP
ENABLE_SENDMAIL
```

This option also accepts any number of arguments. When these are given, each argument is treated as a name of a build flag. `Mailutils-config` checks if such a flag was defined and prints its full name if so. It exits with zero code if all the flags given on the command line are defined. Otherwise, it exits with code of 1.

The comparison of the flag names is case-insensitive. The arguments given need not include the leading prefix (i.e. the characters up to and including the first underscore character).

Given the previous example, the invocation

```
$ mailutils --info readline use_libpam pop
```

will produce the following output:

```
WITH_READLINE
USE_LIBPAM
ENABLE_POP
```

and will exit with a zero status.

The following command:

```
$ mailutils --info readline gssapi pop
```

will exit with status 1, and will print:

```
WITH_READLINE
ENABLE_POP
```

since `WITH_GSSAPI` flag is not defined.

The flags and their meanings are:

USE_LIBPAM
: The Mailutils uses PAM libraries.

HAVE_LIBLTDL
: The GNU wrapper library '`libltdl`' is present and is used by Mailutils. See section "Using libltdl" in *Using libltdl*, for more information on '`libltdl`' library.

WITH_BDB2
: Support for Berkeley DB is compiled in (the package was configured with '`--with-db2`' option).

WITH_NDBM
: Support for NDBM is compiled in (the package was configured with '`--with-ndbm`' option).

WITH_OLD_DBM
: Support for old style DBM is compiled in (the package was configured with '`--with-dbm`' option).

WITH_GDBM
: Support for GNU DBM is compiled in (the package was configured with '`--with-gdbm`' option). See section "Introduction" in *The GNU DBM Manual*, for more information about this library.

WITH_GNUTLS
: Support for GnuTLS (a Transport Layer Security Library) is compiled in (the package was configured with '`--with-gnutls`' option).

WITH_GSASL
: Support for GNU SASL is compiled in (the package was configured with '`--with-gsasl`' option). See section "Introduction" in *The GNU SASL Manual*, for more information about this library.

WITH_GSSAPI
: Support for GSSAPI is compiled in (the package was configured with '`--with-gssapi`' option).

WITH_GUILE
: Support for Guile extension language is built (the package was configured with '`--with-guile`' option). See section "Overview" in *The Guile Reference Manual*, for more information about Guile.

WITH_PTHREAD
: The POSIX thread support is compiled in.

WITH_READLINE
: The readline support is enabled (the package was configured with '`--with-readline`' option). See section "Top" in *The GNU Readline Library*, for more information.

HAVE_MYSQL
: Authentication via MySQL is supported (the package was configured with '`--enable-mysql`' option).

ENABLE_VIRTUAL_DOMAINS
: Support for mail virtual domains is enabled (the package was configured with '`--enable-virtual-domains`' option).

ENABLE_IMAP
: Support for IMAP4 protocol is enabled.

ENABLE_POP
: Support for POP3 protocol is enabled.

ENABLE_MH
: Support for mailboxes in MH format is enabled.

ENABLE_MAILDIR
: Support for mailboxes in MAILDIR format is enabled.

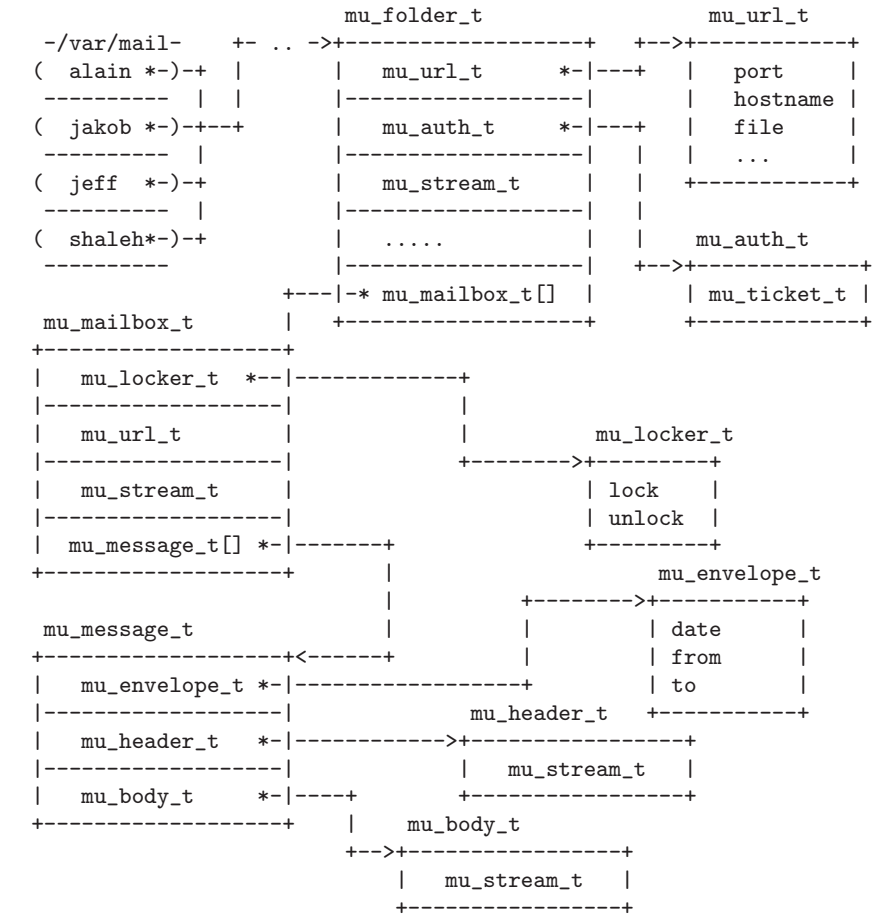
ENABLE_SMTP
: Support for SMTP mailer is enabled.

ENABLE_SENDMAIL
: Support for Sendmail mailer is enabled.

3 Mailutils Libraries

3.1 Framework

Wherever the mail is and whatever format it is stored in, it is operated upon using the same set of functions. To unified the C API, GNU Mailutils offers a heteroclite set of objects that work in aggregation to do operations on emails. Each object does a specific task and delegates non-related tasks to others. The object comes alive by specifying a *URL* parameter when created, it will indicate the storage format or protocol (POP3, IMAP4, MH, MAILDIR, etc ..).



As an example, here is a simplified version of `from` command. It lists the '`From`' and '`Subject`' headers of every mail in a mailbox.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <mailutils/mailutils.h>

int
```

```
main (int argc, const char **argv)
{
  char *from;
  char *subject;
  mu_mailbox_t mbox;
  size_t msgno, total = 0;
  int status;

  /* Register the formats. */
  mu_register_all_mbox_formats ();

  status = mu_mailbox_create_default (&mbox, argv[1]);
  if (status != 0)
    {
      mu_error ("mu_mailbox_create: %s", mu_strerror (status));
      exit (EXIT_FAILURE);
    }

  status = mu_mailbox_open (mbox, MU_STREAM_READ);
  if (status != 0)
    {
      mu_error ("mu_mailbox_open: %s", mu_strerror (status));
      exit (EXIT_FAILURE);
    }

  mu_mailbox_messages_count (mbox, &total);

  for (msgno = 1; msgno <= total; msgno++)
    {
      mu_message_t msg;
      mu_header_t hdr;

      if ((status = mu_mailbox_get_message (mbox, msgno, &msg)) != 0
          || (status = mu_message_get_header (msg, &hdr)) != 0)
        {
          mu_error ("Error message: %s", mu_strerror (status));
          exit (EXIT_FAILURE);
        }

      if (mu_header_aget_value (hdr, MU_HEADER_FROM, &from))
        from = strdup ("(NO FROM)");

      if (mu_header_aget_value (hdr, MU_HEADER_SUBJECT, &subject))
        subject = strdup ("(NO SUBJECT)");

      printf ("%s\t%s\n", from, subject);
      free (from);
      free (subject);
    }

  status = mu_mailbox_close (mbox);
  if (status != 0)
    {
      mu_error ("mu_mailbox_close: %s", mu_strerror (status));
      exit (EXIT_FAILURE);
    }

  mu_mailbox_destroy (&mbox);
```

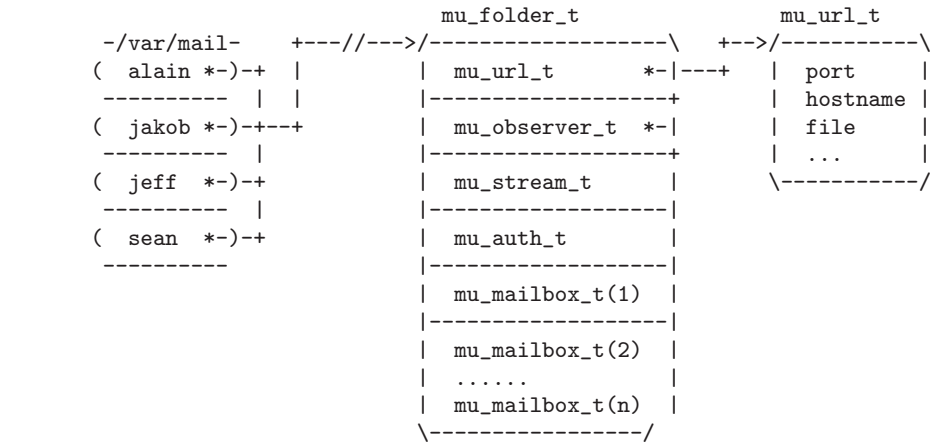
```
  return 0;
}
```

Here is a sample output produced by this program:

```
% ./sfrom pop://alain@localhost
Passwd: xxxx
Jim Meyering <meyering@foo.org>      fetish(shellutils) beta
François Pinard <pinard@bar.org> recode new alpha
...
```

3.1.1 Folder

```
/* Prefix mu_folder_ is reserved. */
#include <mailutils/folder.h>
```



Data structures:

```
struct mu_list_response
{
  int type;
  int separator;
  char *name;
};
```

`int mu_folder_create` (*mu_folder_t *, const char **`url`) [Function]

`void mu_folder_destroy` (*mu_folder_t **) [Function]

`int mu_folder_open` (*mu_folder_t, int* `flag`) [Function]

`int mu_folder_close` (*mu_folder_t*) [Function]

`int mu_folder_delete` (*mu_folder_t, const char **`mailbox`) [Function]

`int mu_folder_rename` (*mu_folder_t, const char *, const char **`mailbox`) [Function]

`int mu_folder_subscribe` (*mu_folder_t, const char **`mailbox`) [Function]

`int mu_folder_unsubscribe` (*mu_folder_t, const char **`mailbox`) [Function]

`int mu_folder_list` (*mu_folder_t*, *const char* ***ref***, *const char* ***wcard***, *size_t* ***size***, *mu_list_t* ***list***) [Function]

`int mu_folder_lsub` (*mu_folder_t*, *const char* ***ref***, *const char* ***wcard***, *mu_list_t* ***list***) [Function]

`int mu_folder_get_stream` (*mu_folder_t*, *mu_stream_t* *) [Function]

`int mu_folder_set_stream` (*mu_folder_t*, *mu_stream_t*) [Function]

`int mu_folder_get_observable` (*mu_folder_t*, *mu_observable_t* *) [Function]

`int mu_folder_has_debug` (*mu_folder_t*) [Function]

`int mu_folder_get_debug` (*mu_folder_t*, *mu_debug_t* *) [Function]

`int mu_folder_set_debug` (*mu_folder_t*, *mu_debug_t*) [Function]

`int mu_folder_get_authority` (*mu_folder_t*, *mu_authority_t* *) [Function]

`int mu_folder_set_authority` (*mu_folder_t*, *mu_authority_t*) [Function]

`int mu_folder_get_url` (*mu_folder_t*, *mu_url_t* *) [Function]

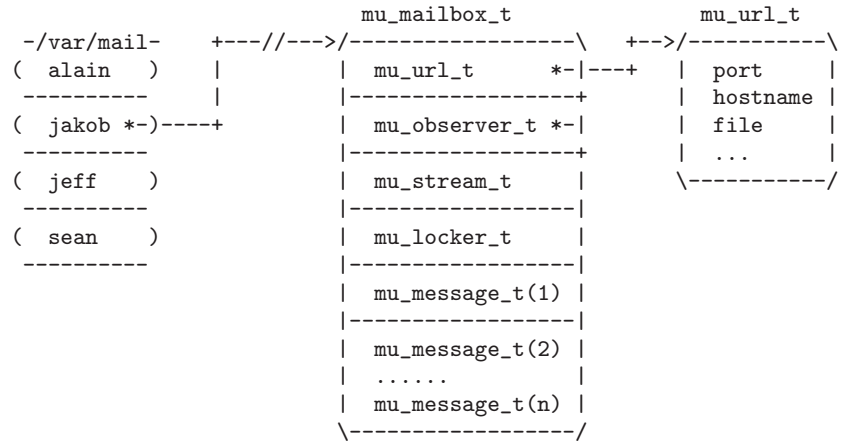
`int mu_folder_set_url` (*mu_folder_t*, *mu_url_t*) [Function]

3.1.2 Mailbox

```
/* Prefix mu_mailbox_ is reserved. */
#include <mailutils/mailbox.h>
```

`mu_mailbox_t` [Data Type]

The `mu_mailbox_t` object is used to hold information and it is an opaque data structure to the user. Functions are provided to retrieve information from the data structure.



`int mu_mailbox_create` (*mu_mailbox_t* ***mbox***, *const char* ***name***) [Function]

The function `mu_mailbox_create` allocates and initializes *mbox*. The concrete mailbox type instantiate is based on the scheme of the url *name*.

The return value is `0` on success and a code number on error conditions:

`MU_ERR_OUT_PTR_NULL`

The pointer *mbox* supplied is `NULL`.

`MU_ERR_NO_HANDLER`
The url *name* supplied is invalid or not supported.

`EINVAL`

`ENOMEM` Not enough memory to allocate resources.

`int mu_mailbox_create_default (`*mu_mailbox_t* ***mbox**, *const char* ***name**`)` [Function]
Create a mailbox with `mu_mailbox_create()` based on the environment variable `MAIL` or the string formed by *_PATH_MAILDIR/user*`"` or `LOGNAME` if *user* is null,

`void mu_mailbox_destroy (`*mu_mailbox_t* ***mbox**`)` [Function]
Destroys and releases resources held by *mbox*.

`int mu_mailbox_open (`*mu_mailbox_t* **mbox**, *int* **flag**`)` [Function]
A connection is open, if no stream was provided, a stream is created based on the *mbox* type. The *flag* can be OR'ed. See `stream_create()` for *flag*'s description.

The return value is `0` on success and a code number on error conditions:

`EAGAIN`
`EINPROGRESS`
Operation in progress.

`EBUSY` Resource busy.

`MU_ERROR_INVALID_PARAMETER`
mbox is `NULL` or flag is invalid.

`ENOMEM` Not enough memory.

`int mu_mailbox_close (`*mu_mailbox_t* **mbox**`)` [Function]
The stream attach to *mbox* is closed.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
mbox is `NULL`.

`int mu_mailbox_flush (`*mu_mailbox_t* **mbox**, *int* **expunge**`)` [Function]

`int mu_mailbox_get_folder (`*mu_mailbox_t* **mbox**, *folder_t* ***folder**`)` [Function]
Get the *folder*.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
mbox is `NULL`.

`int mu_mailbox_set_folder (`*mu_mailbox_t* **mbox**, *mu_folder_t* **folder**`)` [Function]

`int mu_mailbox_uidvalidity (`*mu_mailbox_t* **mbox**, *unsigned long* ***number**`);` [Function]
Give the uid validity of *mbox*.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
mbox is `NULL`.

`int mu_mailbox_uidnext` (*mu_mailbox_t* **`mbox`**, *size_t* ***`number`**); [Function]
Give the next predicted uid for *mbox*.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
mbox is `NULL`.

`int mu_mailbox_get_message` (*mu_mailbox_t* **`mbox`**, *size_t* **`msgno`**, *mu_message_t* ***`message`**) [Function]
Retrieve message number *msgno*, *message* is allocated and initialized.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
mbox is `NULL` or *msgno* is invalid.

`ENOMEM` Not enough memory.

`int mu_mailbox_append_message` (*mu_mailbox_t* **`mbox`**, *mu_message_t* **`message`**) [Function]
The *message* is appended to the mailbox *mbox*.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
mbox is `NULL` or *message* is invalid.

`int mu_mailbox_messages_count` (*mu_mailbox_t* **`mbox`**, *size_t* ***`number`**); [Function]
Give the number of messages in *mbox*.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
mbox is `NULL`.

`int mu_mailbox_messages_recent` (*mu_mailbox_t* **`mbox`**, *size_t* ***`number`**); [Function]
Give the number of recent messages in *mbox*.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
mbox is `NULL`.

`int mu_mailbox_message_unseen` (*mu_mailbox_t* **`mbox`**, *size_t* ***`number`**); [Function]
Give the number of first unseen message in *mbox*.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
mbox is `NULL`.

`int mu_mailbox_expunge` (*mu_mailbox_t* **mbox**) [Function]
All messages marked for deletion are removed.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
mbox is `NULL`.

`int mu_mailbox_save_attributes` (*mu_mailbox_t* **mbox**) [Function]

`int mu_mailbox_get_size` (*mu_mailbox_t* **mbox**, *mu_off_t* ***size**) [Function]
Gives the *mbox* size.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
mbox is `NULL`.

`int mu_mailbox_is_updated` (*mu_mailbox_t* **mbox**) [Function]

`int mu_mailbox_scan` (*mu_mailbox_t* **mbox**, *size_t* **msgno**, *size_t* ***count**); [Function]
Scan the mailbox for new messages starting at message *msgno*.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
mbox is `NULL`.

`ENOMEM` Not enough memory.

`int mu_mailbox_get_stream` (*mu_mailbox_t* **mbox**, *mu_stream_t* ***stream**) [Function]
The mailbox stream is put in *stream*.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
mbox is invalid or *stream* is `NULL`.

`int mu_mailbox_set_stream` (*mu_mailbox_t* **mbox**, *mu_stream_t* **stream**) [Function]
Set the *stream* connection to use for the mailbox.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
mbox or *stream* is `NULL`.

`int mu_mailbox_get_locker` (*mu_mailbox_t* **mbox**, *mu_locker_t* ***locker**) [Function]
Get the *mu_locker_t* object.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
mbox is `NULL`.

`int mu_mailbox_set_locker (`*mu_mailbox_t* **`mbox`**`, `*mu_locker_t* `locker``)` [Function]

Set the type of locking done by the *mbox*.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is `NULL`.

`int mu_mailbox_get_property (`*mu_mailbox_t* **`mbox`**`, `*mu_property_t* `*property``)` [Function]

Get the property object. The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is `NULL`.

`ENOMEM`

`int mu_mailbox_get_url (`*mu_mailbox_t* **`mbox`**`, `*mu_url_t* `*url``)` [Function]

Gives the constructed *url*.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is `NULL`.

`int mu_mailbox_has_debug (`*mu_mailbox_t* **`mbox`**`)` [Function]

`int mu_mailbox_get_debug (`*mu_mailbox_t* **`mbox`**`, `*mu_debug_t* `*debug``)` [Function]

Get a debug object. The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is `NULL`.

`ENOMEM`

`int mu_mailbox_set_debug (`*mu_mailbox_t* **`mbox`**`, `*mu_debug_t* **`debug`**`)` [Function]

`int mu_mailbox_get_observable (`*mu_mailbox_t mbox* **`mbox`**`,` [Function]
mu_observable_t `*observable``)`

Get the observable object.

The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is `NULL`.

`ENOMEM`
: Not enough memory.

`int mu_mailbox_lock (`*mu_mailbox_t* **`mbox`**`)` [Function]

`int mu_mailbox_unlock (`*mu_mailbox_t* **`mbox`**`)` [Function]

3.1.3 Mailer

```
/* Prefix mu_mailer_ is reserved. */
#include <mailutils/mailer.h>
```

`int mu_mailer_create` (*mu_mailer_t *, const char **`url`) [Function]

`void mu_mailer_destroy` (*mu_mailer_t **) [Function]

`int mu_mailer_open` (*mu_mailer_t, int* `flags`) [Function]

`int mu_mailer_close` (*mu_mailer_t*) [Function]

`int mu_mailer_send_message` (*mu_mailer_t* `mailer`, *mu_message_t* `msg`, *mu_address_t* `from`, *mu_address_t* `to`); [Function]

If *from* is not `NULL`, it must contain a single fully qualified RFC2822 email address which will be used as the envelope from address. This is the address to which delivery status notifications are sent, so it never matters what it is set to until it **really** matters. This is equivalent to Sendmail's '`-f`' flag.

The default for *from* is provided by the specific mailer.

If to is not `NULL`, then the message will be sent to the list of addresses that it specifies.

The default for *to* is to use the contents of the standard "To:", "Cc:", and "Bcc:" fields, this is equivalent to Sendmail's '`-t`' flag.

`int mu_mailer_get_property` (*mu_mailer_t, mu_property_t **) [Function]

`int mu_mailer_get_stream` (*mu_mailer_t, mu_stream_t **) [Function]

`int mu_mailer_set_stream` (*mu_mailer_t, mu_stream_t*) [Function]

`int mu_mailer_get_debug` (*mu_mailer_t, mu_debug_t **) [Function]

`int mu_mailer_set_debug` (*mu_mailer_t, mu_debug_t*) [Function]

`int mu_mailer_get_observable` (*mu_mailer_t, observable_t **) [Function]

`int mu_mailer_get_url` (*mu_mailer_t, url_t **) [Function]

`int mu_mailer_check_from` (*mu_address_t* `from`) [Function]

`int mu_mailer_check_to` (*mu_address_t* `to`) [Function]

Some Usage Notes

Some possible use cases the API must support are:

- original submission
  0. fill in header addresses
  1. `mu_mailer_send_message(mailer, msg, NULL, NULL)`
     - from will be filled in if missing,
     - Bcc's will be deleted before delivery to a non-bcc address,
     - message-id and date will be added, if missing,
     - a `To:` or `Apparently-To:` header will be added if non is present (for RFC compliance)

- MTA-style '`.forward`' (and Sieve-style redirect)
  1. get the envelope from of the message to be forwarded
  2. `mu_mailer_send_message(mailer, msg, from, to)`
- MUA-style bounce
  1. add `Resent-[To,From,...]`
  2. `mu_mailer_send_message(mailer, msg, NULL, to)`
- DSN "bounce"
  1. compose DSN
  2. `mu_mailer_deliver(mailer, msg, address_t("<>"), to)`

     Don't want mail loops, so the null but valid SMTP address of '`<>`' is the envelope From.

The Sendmail Mailer

'`/sbin/sendmail`' isn't always Sendmail... Sometimes it's a Sendmail-compatible wrapper, so assume '`/sbin/sendmail`' understands only a recipient list, '`-f`' and '`-oi`', these seem to be pretty basic. Cross fingers.

Pipe to "/sbin/sendmail -oi [-f *from*] [*to*...]", supplying '`-f`' if there was a from, and supplying the recipient list from the to (if there is no recipient list, assume it will read the message contents for the recipients).

Caution: since the `stdout` and `stderr` of Sendmail is closed, we have no way of ever giving feedback on failure. Also, what should the return code be from `mu_mailer_send_message()` when Sendmail returns '`1`'? '`1`' maps to `EPERM`, which is less than descriptive!

The SMTP Mailer

This mailer does **not** canonicalize the message. This must be done before sending the message, or it may be assumed that the MTA will do so.

It does blind out the Bcc: header before sending, though.

Caution: Mutt always puts the recipient addresses on the command line, even Bcc: ones, do we strip the Bcc: before forwarding with SMTP?

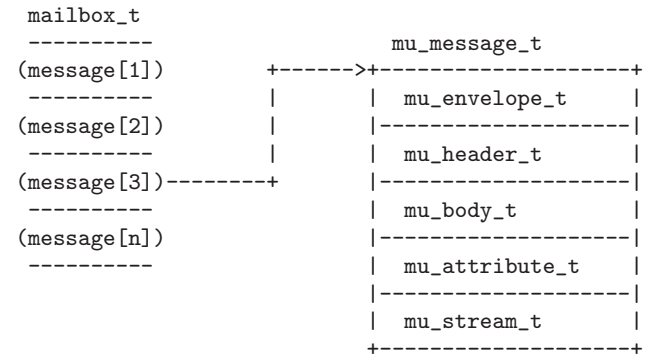
Non-RFC822 Addresses

An address that has no domain is not and RFC822 email address. What do I do with them? Should the user of the API be responsible for determining what is mean by email to "John" means? Or should the be able to configure Sendmail to decide globally what this means. If so, we can pass the address to Sendmail, but we have to decide for SMTP! So, right now these addresses are rejected. This could be changed.

3.1.4 Message

```
/* Prefix mu_message_ is reserved. */
#include <mailutils/message.h>
```

The `mu_message_t` object is a convenient way to manipulate messages. It encapsulates the `envelope_t`, the `header_t` and the `body_t`.



`void mu_message_create` (*mu_message_t* ***msg***, *void* ***owner***) [Function]

`void mu_message_destroy` (*mu_message_t* ***msg***, *void* ***owner***) [Function]
The resources allocate for *msg* are freed.

`int mu_message_create_copy` (*mu_message_t* ***to***, *mu_message_t* ***from***) [Function]

`void* mu_message_get_owner` (*mu_message_t* ***msg***) [Function]

`int mu_message_is_modified` (*mu_message_t* ***msg***) [Function]

`int mu_message_clear_modified` (*mu_message_t* ***msg***) [Function]

`int mu_message_get_mailbox` (*mu_message_t* ***msg***, *mu_mailbox_t* ***mbox***) [Function]

`int mu_message_set_mailbox` (*mu_message_t* ***msg***, *mu_mailbox_t* ***mbox***, *void* ***owner***) [Function]

`int mu_message_ref` (*mu_message_t* ***msg***) [Function]

`int mu_message_get_envelope` (*mu_message_t* ***msg***, *mu_envelope_t* ***envelope***) [Function]

`int mu_message_set_envelope` (*mu_message_t* ***msg***, *mu_envelope_t* ***envelope***, *void* ***owner***) [Function]

`int mu_message_get_header` (*mu_message_t* ***msg***, *mu_header_t* ***header***) [Function]
Retrieve *msg* header.

`int mu_message_set_header` (*mu_message_t* ***msg***, *mu_header_t* ***header***, *void* ***owner***) [Function]

`int mu_message_get_body` (*mu_message_t* ***msg***, *mu_body_t* ***body***) [Function]

`int mu_message_set_body` (*mu_message_t* ***msg***, *mu_body_t* ***body***, *void* ***owner***) [Function]

`int mu_message_get_stream` (*mu_message_t* ***msg***, *mu_stream_t* ***stream***) [Function]

`int mu_message_set_stream` (*mu_message_t* ***msg***, *mu_stream_t* ***stream***, *void* ***owner***) [Function]

`int mu_message_get_attribute (`*mu_message_t* **`msg`**, *mu_attribute_t* **`*attribute`**`)` [Function]

`int mu_message_set_attribute (`*mu_message_t* **`msg`**, *mu_attribute_t* **`attribute`**, *void* ***`owner`**`)` [Function]

`int mu_message_get_observable (`*mu_message_t* **`msg`**, *mu_observable_t* **`*observable`**`)` [Function]

`int mu_message_is_multipart (`*mu_message_t* **`msg`**, *int* ***`multi`**`)` [Function]
Set **multi* to non-zero value if *msg* is multi-part.

`int mu_message_set_is_multipart (`*mu_message_t* **`msg`**, *int* `(`***`_is_multipart`**`)` *(mu_message_t, int *), void *);* [Function]

`int mu_message_size (`*mu_message_t* **`msg`**, *size_t* ***`size`**`)` [Function]

`int mu_message_set_size (`*mu_message_t* **`msg`**, *int* `(`***`_size`**`)` *(mu_message_t, size_t *), void* ***`owner`**`)` [Function]

`int mu_message_lines (`*mu_message_t* **`msg`**, *size_t* ***`size`**`)` [Function]

`int mu_message_set_lines (`*mu_message_t* **`msg`**, *int* `(`***`_lines`**`)` *(mu_message_t, size_t *), void* ***`owner`**`)` [Function]

`int mu_message_get_num_parts (`*mu_message_t* **`msg`**, *size_t* ***`nparts`**`)` [Function]

`int mu_message_set_get_num_parts (`*mu_message_t* **`msg`**, *int* `(`***`_get_num_parts`**`)` *(mu_message_t, size_t *), void* ***`owner`**`)` [Function]

`int mu_message_get_part (`*mu_message_t* **`msg`**, *size_t* **`part`**, *mu_message_t* ***`msg`**`)` [Function]

`int mu_message_set_get_part (`*mu_message_t* **`msg`**, *int* `(`***`_get_part`**`)` *(mu_message_t, size_t, mu_message_t *), void* ***`owner`**`)` [Function]

`int mu_message_get_uidl (`*mu_message_t* **`msg`**, *char* ***`buffer`**, *size_t* **`buflen`**, *size_t* ***`writen`**`)` [Function]

`int mu_message_set_uidl (`*mu_message_t* **`msg`**, *int* `(`***`_get_uidl`**`)` *(mu_message_t, char *, size_t, size_t *), void* ***`owner`**`)` [Function]

`int mu_message_get_uid (`*mu_message_t* **`msg`**, *size_t* ***`uid`**`)` [Function]

`int mu_message_set_uid (`*mu_message_t* **`msg`**, *int* `(`***`_get_uid`**`)` *(mu_message_t, size_t *), void* ***`owner`**`)` [Function]

`int mu_message_create_attachment (`*const char* ***`content_type`**, *const char* ***`encoding`**, *const char* ***`filename`**, *mu_message_t* ***`newmsg`**`)` [Function]

`int mu_message_save_attachment (`*mu_message_t* **`msg`**, *const char* ***`filename`**, *void* ****`data`**`)` [Function]

`int mu_message_encapsulate (`*mu_message_t* **`msg`**, *mu_message_t* ***`newmsg`**, *void* ****`data`**`)` [Function]

`int mu_message_unencapsulate (`*mu_message_t* **`msg`**, *mu_message_t* ***`newmsg`**, *void* ****`data`**`);` [Function]

`int mu_message_get_attachment_name` (*mu_message_t* **msg**, *char *name*, *size_t* **bufsize**, *size_t *size*); [Function]

`int mu_message_aget_attachment_name` (*mu_message_t* **msg**, *char **name*); [Function]

`int mu_message_save_to_mailbox` (*mu_message_t* **msg**, *mu_ticket_t* **ticket**, *mu_debug_t* **debug**, *const char *toname*); [Function]

3.1.5 Envelope

```
/* Prefix mu_envelope_ is reserved. */
#include <mailutils/envelope.h>
```

`int mu_envelope_create` (*mu_envelope_t *, void **) [Function]
Primarily for internal use.

`void mu_envelope_destroy` (*mu_envelope_t *, void **) [Function]
Primarily for internal use.

`void* mu_envelope_get_owner` (*mu_envelope_t*) [Function]

`int mu_envelope_sender` (*mu_envelope_t*, *char *, size_t, size_t **) [Function]
Get the address that this message was reportedly received from. This would be the "mail from" argument if the message was delivered or received via SMTP, for example.

`int mu_envelope_set_sender` (*mu_envelope_t*, *int* (**_sender*) (*mu_envelope_t*, *char *, size_t, size_t **), *void **) [Function]
Primarily for internal use. The implementation of `mu_envelope_t` depends on the mailbox type, this allows the function which actually gets the sender to be set by the creator of an `mu_envelope_t`.

`int mu_envelope_date` (*mu_envelope_t*, *char *, size_t, size_t **) [Function]
Get the date that the message was delivered to the mailbox, in something close to ANSI `ctime()` format: Mon Jul 05 13:08:27 1999.

`int mu_envelope_set_date` (*mu_envelope_t*, *int* (**_date*) (*mu_envelope_t*, *char *, size_t, size_t **), *void **) [Function]
Primarily for internal use. The implementation of `mu_envelope_t` depends on the mailbox type, this allows the function which actually gets the date to be set by the creator of an `mu_envelope_t`.

3.1.6 Headers

```
/* Prefix mu_header_ is reserved. */
#include <mailutils/header.h>
```

So far we plan support for RFC822 and plan for RFC1522. With RFC1522 non-ASCII characters will be encoded.

`int mu_header_create` (*mu_header_t *hdr*, *const char *blurb*, *size_t* **len**, *void *owner*) [Function]
Initialize a *hdr* to a supported type. If *blurb* is not `NULL`, it is parsed.

`void mu_header_destroy` (*mu_header_t* ***hdr***, *void* ***owner***) [Function]
The resources allocated for *hdr* are freed.

`void* mu_header_get_owner` (*mu_header_t* ***hdr***) [Function]

`int mu_header_is_modified` (*mu_header_t* ***hdr***) [Function]

`int mu_header_clear_modified` (*mu_header_t* ***hdr***) [Function]

`int mu_header_set_value` (*mu_header_t* ***hdr***, *const char* ***fn***, *const char* ***fv***, *int* ***n***) [Function]
Some basic macros are already provided for RFC822.

`MU_HEADER_UNIX_FROM`
From

`MU_HEADER_RETURN_PATH`
Return-Path

`MU_HEADER_RECEIVED`
Received

`MU_HEADER_DATE`
Date

`MU_HEADER_FROM`
From

`MU_HEADER_SENDER`
Sender

`MU_HEADER_RESENT_FROM`
Resent-From

`MU_HEADER_SUBJECT`
Subject

`MU_HEADER_SENDER`
Sender

`MU_HEADER_RESENT_SENDER`
Resent-SENDER

`MU_HEADER_TO`
To

`MU_HEADER_RESENT_TO`
Resent-To

`MU_HEADER_CC`
Cc

`MU_HEADER_RESENT_CC`
Resent-Cc

`MU_HEADER_BCC`
Bcc

`MU_HEADER_RESENT_BCC`
: Resent-Bcc

`MU_HEADER_REPLY_TO`
: Reply-To

`MU_HEADER_RESENT_REPLY_TO`
: Resent-Reply-To

`MU_HEADER_MESSAGE_ID`
: Message-ID

`MU_HEADER_RESENT_MESSAGE_ID`
: Resent-Message-ID

`MU_HEADER_IN_REPLY_TO`
: In-Reply-To

`MU_HEADER_REFERENCE`
: Reference

`MU_HEADER_REFERENCES`
: References

`MU_HEADER_ENCRYPTED`
: Encrypted

`MU_HEADER_PRECEDENCE`
: Precedence

`MU_HEADER_STATUS`
: Status

`MU_HEADER_CONTENT_LENGTH`
: Content-Length

`MU_HEADER_CONTENT_LANGUAGE`
: Content-Language

`MU_HEADER_CONTENT_TRANSFER_ENCODING`
: Content-transfer-encoding

`MU_HEADER_CONTENT_ID`
: Content-ID

`MU_HEADER_CONTENT_TYPE`
: Content-Type

`MU_HEADER_CONTENT_DESCRIPTION`
: Content-Description

`MU_HEADER_CONTENT_DISPOSITION`
: Content-Disposition

`MU_HEADER_CONTENT_MD5`
: Content-MD5

`MU_HEADER_MIME_VERSION`
: MIME-Version

`MU_HEADER_X_UIDL`
: X-UIDL

`MU_HEADER_X_UID`
: X-UID

`MU_HEADER_X_IMAPBASE`
: X-IMAPbase

`MU_HEADER_ENV_SENDER`
: X-Envelope-Sender

`MU_HEADER_ENV_DATE`
: X-Envelope-Date

`MU_HEADER_FCC`
: Fcc

`MU_HEADER_DELIVERY_DATE`
: Delivery-date

`MU_HEADER_ENVELOPE_TO`
: Envelope-to

`int mu_header_get_value (`*mu_header_t* **`hdr`**, *const char* ***`fn`**, *char* ***`fv`**, *size_t* **`len`**, *size_t* ***`n`**`)` [Function]

Value of field-name *fn* is returned in buffer *fv* of size *len*. The number of bytes written is put in *n*.

`int mu_header_aget_value (`*mu_header_t* **`hdr`**, *const char* ***`fn`**, *char* ****`fv`**`)` [Function]

The value is allocated.

`int mu_header_get_address (`*mu_header_t* **`hdr`**, *const char* ***`buf`**, *address_t* ***`addr`**`)` [Function]

`int mu_header_get_stream (`*mu_header_t* **`hdr`**, *stream_t* ***`stream`**`)` [Function]

`int mu_header_set_stream (`*mu_header_t* **`hdr`**, *stream_t* **`stream`**, *void* *`)` [Function]

`int mu_header_get_field_count (`*mu_header_t* **`hdr`**, *size_t* ***`count`**`)` [Function]

`int mu_header_get_field_value (`*mu_header_t* **`hdr`**, *size_t* **`index`**, *char* *, *size_t*, *size_t* *`)` [Function]

`int mu_header_get_field_name (`*mu_header_t* **`hdr`**, *size_t* **`index`**, *char* *, *size_t*, *size_t* *`)` [Function]

`int mu_header_aget_field_value (`*mu_header_t* **`hdr`**, *size_t* **`index`**, *char* ****`)` [Function]

`int mu_header_aget_field_name (`*mu_header_t* **`hdr`**, *size_t* **`index`**, *char* ****`)` [Function]

`int mu_header_get_value_unfold` (*mu_header_t* **hdr**, *const char* ***name**, *char* ***buffer**, *size_t* **buflen**, *size_t* ***n**) [Function]

`int mu_header_aget_value_unfold` (*mu_header_t* **hdr**, *const char* ***name**, *char* ****value**) [Function]

`int mu_header_get_field_value_unfold` (*mu_header_t* **hdr**, *size_t* **num**, *char* ***buf**, *size_t* **buflen**, *size_t* ***nwritten**) [Function]

`int mu_header_aget_field_value_unfold` (*mu_header_t* **hdr**, *size_t* **num**, *char* ****value**); [Function]

`int mu_header_size` (*mu_header_t* **hdr**, *size_t* *); [Function]

`int mu_header_lines` (*mu_header_t* **hdr**, *size_t* *); [Function]

`int mu_header_set_set_value` (*mu_header_t* **hdr**, *int* (***_set_value**) (*mu_header_t*, *const char* *, *const char* *, *int*), *void* *); [Function]

`int mu_header_set_get_value` (*mu_header_t* **hdr**, *int* (***_get_value**) (*mu_header_t*, *const char* *, *char* *, *size_t*, *size_t* *), *void* *); [Function]

`int mu_header_set_get_fvalue` (*mu_header_t* **hdr**, *int* (***_get_value**) (*mu_header_t*, *const char* *, *char* *, *size_t*, *size_t* *), *void* *); [Function]

`int mu_header_set_size` (*mu_header_t* **hdr**, *int* (***_size**) (*mu_header_t*, *size_t* *), *void* *); [Function]

`int mu_header_set_lines` (*mu_header_t* **hdr**, *int* (***_lines**) (*mu_header_t*, *size_t* *), *void* *); [Function]

`int mu_header_set_fill` (*mu_header_t* **hdr**, *int* (***_fill**) (*mu_header_t*, *char* *, *size_t*, *mu_off_t*, *size_t* *), *void* ***owner**); [Function]

3.1.7 Body

```
/* Prefix mu_body_ is reserved. */
#include <mailutils/body.h>
```

`int mu_body_create` (*mu_body_t* ***body**, *void* ***owner**) [Function]
Initialize an object *body*.

`void mu_body_destroy` (*mu_body_t* ***body**) [Function]
The resources allocated are release.

`void* mu_body_get_owner` (*mu_body_t* **body**) [Function]

`int mu_body_is_modified` (*mu_body_t* **body**) [Function]

`int mu_body_clear_modified` (*mu_body_t* **body**) [Function]

`int mu_body_get_stream` (*mu_body_t* **body**, *stream_t* ***stream**) [Function]

`int mu_body_set_stream` (*mu_body_t* **body**, *stream_t* **stream**, *void* ***owner**) [Function]

`int mu_body_get_filename` (*mu_body_t* **body**, *char* ***buffer**, *size_t* **buflen**, *size_t* ***writen**) [Function]

`int mu_body_size` (*mu_body_t* **body**, *size_t* ***size**) [Function]

`int mu_body_set_size` (*mu_body_t* **body**, *int* (***_size**) (*mu_body_t*, *size_t* *), *void* ***owner**) [Function]

`int mu_body_lines` (*mu_body_t* **body**, *size_t* ***lines**) [Function]

`int mu_body_set_lines` (*mu_body_t* **body**, *int* (***_lines**) (*mu_body_t*, *size_t* *), *void* ***owner**) [Function]

3.1.8 Attribute

```
/* Prefix mu_attribute_ is reserved. */
#include <mailutils/attribute.h>
```

`int mu_attribute_create` (*mu_attribute_t* ***attr**, *void* *) [Function]

`void mu_attribute_destroy` (*mu_attribute_t* ***attr**, *void* *) [Function]

`void* mu_attribute_get_owner` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_is_modified` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_clear_modified` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_set_modified` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_is_userflag` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_is_seen` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_is_answered` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_is_flagged` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_is_deleted` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_is_draft` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_is_recent` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_is_read` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_set_userflag` (*mu_attribute_t* **attr**, *int*) [Function]

`int mu_attribute_set_seen` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_set_answered` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_set_flagged` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_set_deleted` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_set_draft` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_set_recent` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_set_read` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_unset_userflag` (*mu_attribute_t* **attr**, *int*) [Function]

`int mu_attribute_unset_seen` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_unset_answered` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_unset_flagged` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_unset_deleted` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_unset_draft` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_unset_recent` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_unset_read` (*mu_attribute_t* **attr**) [Function]

`int mu_attribute_get_flags` (*mu_attribute_t* **attr**, *int* *) [Function]

`int mu_attribute_set_flags` (*mu_attribute_t* **attr**, *int*) [Function]

`int mu_attribute_unset_flags` (*mu_attribute_t* **attr**, *int*) [Function]

`int mu_attribute_set_set_flags` (*mu_attribute_t* **attr**, *int* (*_set_flags) (*mu_attribute_t*, *int*), *void* *) [Function]

`int mu_attribute_set_unset_flags` (*mu_attribute_t* **attr**, *int* (*_unset_flags) (*mu_attribute_t*, *int*), *void* *) [Function]

`int mu_attribute_set_get_flags` (*mu_attribute_t* **attr**, *int* (*_get_flags) (*mu_attribute_t*, *int* *), *void* *) [Function]

`int mu_attribute_is_equal` (*mu_attribute_t* **attr1**, *mu_attribute_t* attr2) [Function]

`int mu_attribute_copy` (*mu_attribute_t* **dst**, *mu_attribute_t* **src**) [Function]

`int mu_attribute_to_string` (*mu_attribute_t* **attr**, *char* ***buf**, *size_t* **len**, *size_t* ***writen**) [Function]

`int string_to_flags` (*const char* ***buf**, *int* *) [Function]

3.1.9 Stream

```
#include <mailutils/stream.h>
```

These generic flags are interpreted as appropriate to the specific streams.

`MU_STREAM_READ`
: The stream is open read only.

`MU_STREAM_WRITE`
: The stream is open write only.

`MU_STREAM_RDWR`
: The stream is open read and write.

`MU_STREAM_APPEND`
: The stream is open in append mode for writing.

`MU_STREAM_CREAT`
: The stream open will create the underlying resource (such as a file) if it doesn't exist already.

`MU_STREAM_NONBLOCK`
: The stream is set non blocking.

`MU_STREAM_NO_CHECK`
: Stream is destroyed without checking for the owner.

`MU_STREAM_SEEKABLE`

`MU_STREAM_NO_CLOSE`
: Stream doesn't close it's underlying resource when it is closed or destroyed.

`MU_STREAM_ALLOW_LINKS`

`int mu_file_stream_create (`*mu_stream_t* ***stream**, *const char* ***filename**, *int* **flags**`)` [Function]

`int mu_tcp_stream_create (`*mu_stream_t* ***stream**, *const char* ***host**, *int* **port**, *int* **flags**`)` [Function]

`int mu_mapfile_stream_create (`*mu_stream_t* ***stream**, *const char* ***filename**, *int* **flags**`)` [Function]

`int mu_memory_stream_create (`*mu_stream_t* ***stream**, *const char* ***filename**, *int* **flags**`)` [Function]

`int mu_encoder_stream_create (`*mu_stream_t* ***stream**, *mu_stream_t* **iostream**, *const char* ***encoding**`)` [Function]

`int mu_decoder_stream_create (`*mu_stream_t* ***stream**, *mu_stream_t* **iostream**, *const char* ***encoding**`)` [Function]

`int mu_stdio_stream_create (`*mu_stream_t* ***stream**, *FILE* ***stdio**, *int* **flags**`)` [Function]

If `MU_STREAM_NO_CLOSE` is specified, `fclose()` will not be called on *stdio* when the stream is closed.

`int mu_prog_stream_create (`*mu_stream_t* ***stream**, *const char* ***progname**, *int* **flags**`)` [Function]

`int mu_filter_prog_stream_create (`*mu_stream_t* ***stream**, *const char* ***progname**, *mu_stream_t* **input**`)` [Function]

`void mu_stream_destroy (`*mu_stream_t* ***stream**, *void* ***owner**`)` [Function]

`int mu_stream_open (`*mu_stream_t* **stream**`)` [Function]

`int mu_stream_close (`*mu_stream_t* **stream**`)` [Function]

`int mu_stream_is_seekable (`*mu_stream_t* **stream**`)` [Function]

`int mu_stream_get_fd (`*mu_stream_t* **stream**, *int* ***fd**`)` [Function]

`int mu_stream_get_fd2 (`*mu_stream_t* **stream**, *int* ***fd1**, *int* ***fd2**`)` [Function]

`int mu_stream_read (`*mu_stream_t* **stream**, *char* ***buffer**, *size_t* **buflen**, *mu_off_t* **offset**, *size_t* ***writen**`)` [Function]

`int mu_stream_readline (`*mu_stream_t* **stream**, *char* ***buffer**, *size_t* **buflen**, *mu_off_t* **offset**, *size_t* ***writen**`)` [Function]

`int mu_stream_size (`*mu_stream_t* **stream**, *mu_off_t* ***size**`)` [Function]

`n int` *mu_stream_truncate* `(`*mu_stream_t* **stream**, *mu_off_t* **size**`)` [Function]

`int mu_stream_write` (*mu_stream_t* **stream**, *const char* ***buffer**, *size_t* **buflen**, *mu_off_t* **offset**, *size_t* ***writen**) [Function]

`int mu_stream_setbufsiz` (*mu_stream_t* **stream**, *size_t* **size**) [Function]

`int mu_stream_flush` (*mu_stream_t* **stream**) [Function]

`int mu_stream_create` (*mu_stream_t* ***stream**, *int* **flags**, *void* ***owner**) [Function]

Used to implement a new kind of stream.

`void* mu_stream_get_owner` (*mu_stream_t* **stream**) [Function]

`void mu_stream_set_owner` (*mu_stream_t* **stream**, *void* ***owner**) [Function]

`int mu_stream_get_flags` (*mu_stream_t* **stream**, *int* ***flags**) [Function]

`int mu_stream_set_flags` (*mu_stream_t* **stream**, *int* **flags**) [Function]

`int mu_stream_get_property` (*mu_stream_t* **stream**, *property_t* *) [Function]

`int mu_stream_set_property` (*mu_stream_t* **stream**, *property_t*, *void* *) [Function]

`int mu_stream_get_state` (*mu_stream_t* **stream**, *int* ***state**) [Function]

`MU_STREAM_STATE_OPEN`
: Last action was `mu_stream_open`.

`MU_STREAM_STATE_READ`
: Last action was `mu_stream_read` or `mu_stream_readline`.

`MU_STREAM_STATE_WRITE`
: Last action was `mu_stream_write`.

`MU_STREAM_STATE_CLOSE`
: Last action was `mu_stream_close`.

`int mu_stream_set_destroy` (*mu_stream_t* **stream**, *void* (***_destroy**) (*mu_stream_t*), *void* ***owner**) [Function]

`int mu_stream_set_open` (*mu_stream_t* **stream**, *int* (***_open**) (*mu_stream_t*), *void* ***owner**) [Function]

`int mu_stream_set_close` (*mu_stream_t* **stream**, *int* (***_close**) (*mu_stream_t*), *void* ***owner**) [Function]

`int mu_stream_set_fd` (*mu_stream_t* **stream**, *int* (***_get_fd**) (*mu_stream_t*, *int* *, *int* *), *void* ***owner**) [Function]

`int mu_stream_set_read` (*mu_stream_t* **stream**, *int* (***_read**) (*mu_stream_t*, *char* *, *size_t*, *mu_off_t*, *size_t* *), *void* ***owner**) [Function]

`int mu_stream_set_readline` (*mu_stream_t* **stream**, *int* (***_readline**) (*mu_stream_t*, *char* *, *size_t*, *mu_off_t*, *size_t* *), *void* ***owner**) [Function]

`int mu_stream_set_size` (*mu_stream_t* **stream**, *int* (***_size**) (*mu_stream_t*, *mu_off_t* *), *void* ***owner**) [Function]

`int mu_stream_set_truncate` (*mu_stream_t* **`stream`**, *int* (***`_truncate`**) (*mu_stream_t*, *mu_off_t*), *void* ***`owner`**) [Function]

`int mu_stream_set_write` (*mu_stream_t* **`stream`**, *int* (***`_write`**) (*mu_stream_t*, *const char* *, *size_t*, *mu_off_t*, *size_t* *), *void* ***`owner`**) [Function]

`int mu_stream_set_flush` (*mu_stream_t* **`stream`**, *int* (***`_flush`**) (*mu_stream_t*), *void* ***`owner`**) [Function]

`int mu_stream_set_strerror` (*mu_stream_t* **`stream`**, *int* (***`_fp`**) (*mu_stream_t*, *char* **), *void* ***`owner`**) [Function]

`int mu_stream_sequential_readline` (*mu_stream_ts* **`stream`**, *char* ***`buf`**, *size_t* **`size`**, *size_t* ***`nbytes`**) [Function]

`int mu_stream_sequential_write` (*mu_stream_t* **`stream`**, *char* ***`buf`**, *size_t* **`size`**) [Function]

`int mu_stream_seek` (*mu_stream_t* **`stream`**, *mu_off_t* **`off`**, *int* **`whence`**) [Function]

`int mu_stream_strerror` (*mu_stream_t* **`stream`**, *char* ****`p`**) [Function]

An example using `mu_tcp_stream_create()` to make a simple web client:

```
/* This is an example program to illustrate the use of stream functions.
   It connects to a remote HTTP server and prints the contents of its
   index page */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/time.h>

#include <mailutils/mailutils.h>

char wbuf[1024];
char rbuf[1024];

size_t io_timeout = 3;
size_t io_attempts = 3;

int
http_stream_wait (mu_stream_t stream, int flags, size_t *attempt)
{
  int rc;
  int oflags = flags;
  struct timeval tv;

  while (*attempt < io_attempts)
    {
      tv.tv_sec = io_timeout;
      tv.tv_usec = 0;
      rc = mu_stream_wait (stream, &oflags, &tv);
      switch (rc) {
      case 0:
if (flags & oflags)
  return 0;
```

```
/* FALLTHROUGH */
      case EAGAIN:
      case EINPROGRESS:
+++attempt;
continue;

      default:
return rc;
      }
    }
  return ETIMEDOUT;
}

int
main (int argc, char **argv)
{
  int ret, off = 0;
  mu_stream_t stream;
  size_t nb, size;
  size_t attempt;
  char *url = "www.gnu.org";

  if (argc > 3)
    {
      fprintf (stderr, "usage: %s [hostname [url]]\n", argv[0]);
      exit (1);
    }

  if (argc > 1)
    url = argv[1];

  snprintf (wbuf, sizeof wbuf, "GET %s HTTP/1.0\r\n\r\n",
    argc == 3 ? argv[2] : "/");

  ret = mu_tcp_stream_create (&stream, url, 80, MU_STREAM_NONBLOCK);
  if (ret != 0)
    {
      mu_error ("mu_tcp_stream_create: %s", mu_strerror (ret));
      exit (EXIT_FAILURE);
    }

  for (attempt = 0; (ret = mu_stream_open (stream)); )
    {
      if ((ret == EAGAIN || ret == EINPROGRESS) && attempt < io_attempts)
{
  ret = http_stream_wait(stream, MU_STREAM_READY_WR, &attempt);
  if (ret == 0)
    continue;
}
      mu_error ("mu_stream_open: %s", mu_strerror (ret));
      exit (EXIT_FAILURE);
    }

  for (attempt = 0, size = strlen (wbuf); size > 0; )
    {
      ret = mu_stream_write (stream, wbuf + off, strlen (wbuf), 0, &nb);
      if (ret == 0)
{
```

```
  if (nb == 0)
    {
      mu_error("mu_stream_write: wrote 0 bytes");
      exit (EXIT_FAILURE);
    }
  off += nb;
  size -= nb;
}
      else if (ret == EAGAIN)
        {
  if (attempt < io_attempts)
    {
      ret = http_stream_wait (stream, MU_STREAM_READY_WR, &attempt);
      if (ret)
{
  mu_error ("http_wait failed: %s", mu_strerror (ret));
  return -1;
}
      continue;
    }
  else
    {
      mu_error ("mu_stream_write timed out");
      exit (EXIT_FAILURE);
    }
}
      else
{
  mu_error ("mu_stream_write: %s", mu_strerror (ret));
  exit (EXIT_FAILURE);
}
    }

  attempt = 0;
  for (;;)
    {
      ret = mu_stream_read (stream, rbuf, sizeof (rbuf), 0, &nb);
      if (ret == 0)
{
  if (nb == 0)
    break;
  write (1, rbuf, nb);
}
      else if (ret == EAGAIN)
{
  if (attempt < io_attempts)
    {
      ret = http_stream_wait (stream, MU_STREAM_READY_RD, &attempt);
      if (ret)
{
  mu_error ("http_stream_wait failed: %s", mu_strerror (ret));
  exit (EXIT_FAILURE);
}
    }
  else
    {
      mu_error ("mu_stream_read: %s", mu_strerror (ret));
      exit (EXIT_FAILURE);
```

```
        }
    }
        }

      ret = mu_stream_close (stream);
      if (ret != 0)
        {
          mu_error ("mu_stream_close: %s", mu_strerror (ret));
          exit (EXIT_FAILURE);
        }

      mu_stream_destroy (&stream, NULL);
      exit (EXIT_SUCCESS);
    }
```

3.1.10 Iterator

```
/* Prefix mu_iterator_ is reserved. */
#include <mailutils/iterator.h>
```

`int mu_iterator_create` (*mu_iterator_t* ***iterator***, *void* ***obj***) [Function]

`int mu_iterator_dup` (*mu_iterator_t* ***iterator***, *mu_iterator_t* ***orig***) [Function]

`void mu_iterator_destroy` (*mu_iterator_t* *) [Function]

`int mu_iterator_first` (*mu_iterator_t*) [Function]

`int mu_iterator_next` (*mu_iterator_t*) [Function]

`int mu_iterator_current` (*mu_iterator_t*, *void* ***item*) [Function]

`int mu_iterator_is_done` (*mu_iterator_t*) [Function]

`int mu_iterator_attach` (*mu_iterator_t* ***root***, *mu_iterator_t* ***iterator***) [Function]

`int mu_iterator_detach` (*mu_iterator_t* ***root***, *mu_iterator_t* ***iterator***) [Function]

`void mu_iterator_advance` (*mu_iterator_t* ***iterator***, *void* ***item***) [Function]

`int mu_iterator_set_first` (*mu_iterator_t* ***iterator***, *int* (****first***) (*void* *)) [Function]

`int mu_iterator_set_next` (*mu_iterator_t* ***iterator***, *int* (****next***) (*void* *)) [Function]

`int mu_iterator_set_getitem` (*mu_iterator_t* ***iterator***, *int* (****getitem***) (*void* *, *void* **)) [Function]

`int mu_iterator_set_finished_p` (*mu_iterator_t* ***iterator***, *int* (****finished_p***) (*void* *)) [Function]

`int mu_iterator_set_dup` (*mu_iterator_t* ***itr***, *int* (****dup***) (*void* ***ptr*, *void* ****data***)) [Function]

`int mu_iterator_set_destroy` (*mu_iterator_t* ***itr***, *int* (****destroy***) (*mu_iterator_t* ***itr***, *void* ****data***)) [Function]

`int mu_iterator_set_curitem_p` (*mu_iterator_t* ***itr***, *int* (****curitem_p***) (*void* *, *void* *)) [Function]

3.1.11 Authenticator

```
/* Prefixes mu_authority_, mu_ticket_, and mu_wicket_ are reserved. */
#include <mailutils/auth.h>
```

There are many ways to authenticate to a server. To be flexible the authentication process is provided by three objects `mu_authority_t`, `mu_ticket_t`, and `mu_wicket_t`. The `mu_authority_t` can implement different protocol like APOP, MD5-AUTH, One Time Passwd, etc. By default if a mailbox does not understand or know how to authenticate it falls back to user/passwd authentication. The `mu_ticket_t` is a way for Mailboxes and Mailers provide a way to authenticate when the URL does not contain enough information. The default action is to call the function `mu_authority_authenticate()` which will get the *user* and *passwd* if not set, this function can be overridden by a custom method.

`int mu_ticket_create` (*mu_ticket_t *, void *`owner`*) [Function]

`void mu_ticket_destroy` (*mu_ticket_t *, void *`owner`*) [Function]

`int mu_ticket_set_destroy` (*mu_ticket_t, void* (*) (*mu_ticket_t*), *void *`owner`*) [Function]

`void* mu_ticket_get_owner` (*mu_ticket_t*) [Function]

`int mu_ticket_set_pop` (*mu_ticket_t, int* (*`_pop`) (*mu_ticket_t, url_t, const char *, char ***), *void **) [Function]

`int mu_ticket_pop` (*mu_ticket_t, url_t, const char *, char ***) [Function]

`int mu_ticket_set_data` (*mu_ticket_t, void *, void *`owner`*) [Function]

`int mu_ticket_get_data` (*mu_ticket_t, void ***) [Function]

`int mu_authority_create` (*mu_authority_t *, mu_ticket_t, void **) [Function]

`void mu_authority_destroy` (*mu_authority_t *, void **) [Function]

`void* mu_authority_get_owner` (*mu_authority_t*) [Function]

`int mu_authority_set_ticket` (*mu_authority_t, mu_ticket_t*) [Function]

`int mu_authority_get_ticket` (*mu_authority_t, mu_ticket_t **) [Function]

`int mu_authority_authenticate` (*mu_authority_t*) [Function]

`int mu_authority_set_authenticate` (*mu_authority_t, int* (*`_authenticate`) (*mu_authority_t*), *void **) [Function]

`int mu_authority_create_null` (*mu_authority_t *`authority`, void *`owner`*) [Function]

`int mu_wicket_create` (*mu_wicket_t *, const char **) [Function]

`void mu_wicket_destroy` (*mu_wicket_t **) [Function]

`int mu_wicket_set_filename` (*mu_wicket_t, const char **) [Function]

`int mu_wicket_get_filename` (*mu_wicket_t, char *, size_t, size_t **) [Function]

`int mu_wicket_set_ticket` (*mu_wicket_t, int (*) (mu_wicket_t, const char *, const char *, mu_ticket_t *)*) [Function]

`int mu_wicket_get_ticket` (*mu_wicket_t, mu_ticket_t *, const char *, const char **) [Function]

A simple example of an authenticate function:

```
#include <stdio.h>
#include <string.h>
#include <mailutils/auth.h>

int
my_authenticate (auth_t auth, char **user, char **passwd)
{
  char u[128] = "";
  char p[128] = "";

  /* prompt the user name */
  printf ("User: ");
  fflush (stdout);
  fgets (u, sizeof (u), stdin);
  u[strlen (u) - 1] = '\0'; /* nuke the trailing NL */

  /* prompt the passwd */
  printf ("Passwd: "); fflush (stdout);
  echo_off ();
  fgets (p, sizeof(p), stdin);
  echo_on ();
  p[strlen (p) - 1] = '\0';

  /* duplicate */
  *user = strdup (u);
  *passwd = strdup (p);
  return 0;
}
```

3.1.12 Address

```
/* Prefix address_ is reserved. */
#include <mailutils/address.h>
```

The Internet address format is defined in RFC 822. RFC 822 has been updated, and is now superceeded by RFC 2822, which makes some corrections and clarifications. References to RFC 822 here apply equally to RFC 2822.

The RFC 822 format is more flexible than many people realize, here is a quick summary of the syntax this parser implements, see RFC 822 for the details. '`[]`' pairs mean "optional", '`/`' means "one or the other", and double-quoted characters are literals.

```
addr-spec    = local-part "@" domain
mailbox      = addr-spec ["(" display-name ")"] /
               [display-name] "<" [route] addr-spec ">"
mailbox-list = mailbox ["," mailbox-list]
group        = display-name ":" [mailbox-list] ";"
address      = mailbox / group / unix-mbox
address-list = address ["," address-list]
```

Unix-mbox is a non-standard extension meant to deal with the common practice of using user names as addresses in mail utilities. It allows addresses such as "root" to be parsed correctly. These are **not** valid internet email addresses, they must be qualified before use.

Several address functions have a set of common arguments with consistent semantics, these are described here to avoid repetition.

Since an address-list may contain multiple addresses, they are accessed by a **one-based** index number, *no*. The index is one-based because pop, imap, and other message stores commonly use one-based counts to access messages and attributes of messages.

If *len* is greater than `0` it is the length of the buffer *buf*, and as much of the component as possible will be copied into the buffer. The buffer will be `NULL` terminated.

The size of a particular component may be queried by providing `0` for the *len* of the buffer, in which case the buffer is optional. In this case, if *n* is provided **n* is assigned the length of the component string.

`mu_address_t` [Data Type]

The `mu_address_t` object is used to hold information about a parsed RFC822 address list, and is an opaque data structure to the user. Functions are provided to retrieve information about an address in the address list.

`int mu_address_create` (*mu_address_t* ***addr***, *const char* ****string***) [Function]

This function allocates and initializes *addr* by parsing the RFC822 address-list *string*.

The return value is `0` on success and a code number on error conditions:

`EINVAL` Invalid usage, usually a required argument was `NULL`.

`ENOMEM` Not enough memory to allocate resources.

`ENOENT` Invalid RFC822 syntax, parsing failed.

`int mu_address_createv` (*mu_address_t* ***addr***, *const char* ****sv***, *size_t* ***len***) [Function]

This function allocates and initializes *addr* by parsing the array of pointers to RFC822 address-lists in *sv*. If *len* is `-1`, then *sv* must be `NULL` terminated in the fashion of *argv*, otherwise *len* is the length of the array.

The return value is `0` on success and a code number on error conditions:

`EINVAL` Invalid usage, usually a required argument was `NULL`.

`ENOMEM` Not enough memory to allocate resources.

`ENOENT` Invalid RFC822 syntax, parsing failed.

`void mu_address_destroy` (*mu_address_t* ***addr***) [Function]

The *addr* is destroyed.

`int mu_address_get_nth` (*mu_address_t* **addr**, *size_t* **no**, *mu_address_t* ****ret***) [Function]

`int mu_address_get_email` (*mu_address_t* **addr**, *size_t* **no**, *char** **buf**, *size_t* **len**, *size_t* ***n***) [Function]

Accesses the *no*th email address component of the address list. This address is the plain email address, correctly quoted, suitable for using in an smtp dialog, for example, or as the address part of a contact book entry.

Note that the entry may be valid, but be a group name. In this case success is returned, but the length of the address is `0`.

The return value is `0` on success and a code number on error conditions:

`EINVAL` Invalid usage, usually a required argument was `NULL`.

`ENOENT` The index *no* is outside of the range of available addresses.

`int mu_address_get_local_part (`*mu_address_t* **`addr`**, *size_t* **`no`**, *char* ***`buf`**, *size_t* **`len`**, *size_t* ***`n`**`)` [Function]

Accesses the local-part of an email addr-spec extracted while parsing the *no*th email address.

The return value is `0` on success and a code number on error conditions:

`EINVAL` Invalid usage, usually a required argument was `NULL`.

`ENOENT` The index *no* is outside of the range of available addresses.

`int mu_address_get_domain (`*mu_address_t* **`addr`**, *size_t* **`no`**, *char* ***`buf`**, *size_t* **`len`**, *size_t* ***`n`**`)` [Function]

Accesses the domain of an email addr-spec extracted while parsing the *no*th email address. This will be `0` length for a unix-mbox.

The return value is `0` on success and a code number on error conditions:

`EINVAL` Invalid usage, usually a required argument was `NULL`.

`ENOENT` The index *no* is outside of the range of available addresses.

`int mu_address_get_personal (`*mu_address_t* **`addr`**, *size_t* **`no`**, *char* ***`buf`**, *size_t* **`len`**, *size_t* ***`n`**`)` [Function]

Accesses the display-name describing the *no*th email address. This display-name is optional, so may not be present. If it is not present, but there is an RFC822 comment after the address, that comment will be returned as the personal phrase, as this is a common usage of the comment even though it is not defined in the internet mail standard.

A group is a kind of a special case. It has a display-name, followed by an optional mailbox-list. The display-name will be allocated an address all it's own, but all the other elements (local-part, domain, etc.) will be zero-length. So `"a group: ;"` is valid, will have a count of 1, but `mu_address_get_email()`, and all the rest, will return zero-length output.

The return value is `0` on success and a code number on error conditions:

`EINVAL` Invalid usage, usually a required argument was `NULL`.

`ENOENT` The index *no* is outside of the range of available addresses.

`int mu_address_get_comments (`*mu_address_t* **`addr`**, *size_t* **`no`**, *char* ***`buf`**, *size_t* **`len`**, *size_t* ***`n`**`)` [Function]

Accesses the comments extracted while parsing the *no*th email address. These comments have no defined meaning, and are not currently collected.

The return value is `0` on success and a code number on error conditions:

`EINVAL` Invalid usage, usually a required argument was `NULL`.

`ENOENT` The index *no* is outside of the range of available addresses.

`int mu_address_get_route (`*mu_address_t* **`addr`**, *size_t* **`no`**, *char* ***`buf`**, *size_t* **`len`**, *size_t* ***`n`**`)` [Function]

Accesses the route of an email addr-spec extracted while parsing the *n*oth email address. This is a rarely used RFC822 address syntax, but is legal in SMTP as well. The entire route is returned as a string, those wishing to parse it should look at '`mailutils/parse822.h`'.

The return value is `0` on success and a code number on error conditions:

`EINVAL` Invalid usage, usually a required argument was `NULL`.

`ENOENT` The index *no* is outside of the range of available addresses.

`int mu_address_aget_email (`*mu_address_t* **`addr`**, *size_t* **`no`**, *char* ***`bufp`**`)` [Function]

As above, but mallocs the email address, if present, and write a pointer to it into *bufp*. *bufp* will be `NULL` if there is no email address to return.

The return value is `0` on success and a code number on error conditions:

`EINVAL` Invalid usage, usually a required argument was `NULL`.

`ENOENT` The index *no* is outside of the range of available addresses.

`int mu_address_aget_local_part (`*mu_address_t* **`addr`**, *size_t* **`no`**, *char* ***`buf`**`)` [Function]

`int mu_address_aget_domain (`*mu_address_t* **`addr`**, *size_t* **`no`**, *char* ***`buf`**`)` [Function]

`int mu_address_aget_personal (`*mu_address_t* **`addr`**, *size_t* **`no`**, *char* ***`buf`**`)` [Function]

`int mu_address_is_group (`*mu_address_t* **`addr`**, *size_t* **`no`**, *int* ***`yes`**`)` [Function]

Sets **yes* to `1` if this address is just the name of a group, `0` otherwise. This is faster than checking if the address has a non-zero length personal, and a zero-length local_part and domain.

yes can be `NULL`, though that doesn't serve much purpose other than determining that *no* refers to an address.

Currently, there is no way to determine the end of the group.

The return value is `0` on success and a code number on error conditions:

`EINVAL` Invalid usage, usually a required argument was `NULL`.

`ENOENT` The index *no* is outside of the range of available addresses.

`int mu_address_to_string (`*mu_address_t* ***`addr`**, *char* ***`buf`**, *size_t* **`len`**, *size_t* ***`n`**`)` [Function]

Returns the entire address list as a single RFC822 formatted address list.

The return value is `0` on success and a code number on error conditions:

`EINVAL` Invalid usage, usually a required argument was `NULL`.

`ENOMEM` Not enough memory to allocate resources.

`int mu_address_get_count (`*mu_address_t* **`addr`**, *size_t* ***`count`**`)` [Function]

Returns a count of the addresses in the address list.

If *addr* is `NULL`, the count is `0`. If *count* is not `NULL`, the count will be written to **count*.

The return value is `0`.

`int mu_address_get_group_count (`*mu_address_t* **`addr`**, *size_t* *`)` [Function]

`int mu_address_get_email_count (`*mu_address_t* **`addr`**, *size_t* *`)` [Function]

`int mu_address_get_unix_mailbox_count (`*mu_address_t* **`addr`**, *size_t* *`)` [Function]

`int mu_address_contains_email (`*mu_address_t* **`addr`**, *const char* ***`email`**`)` [Function]

`int mu_address_union (`*mu_address_t* *`a`, *mu_address_t* **`b`**`)` [Function]

`size_t mu_address_format_string (`*mu_address_t* **`addr`**, *char* ***`buf`**, *size_t* **`buflen`**`)` [Function]

Example

```
#include <stdio.h>
#include <string.h>
#include <errno.h>

#include <mailutils/address.h>
#include <mailutils/errno.h>
#include <mailutils/mutil.h>

#define EPARSE MU_ERR_NOENT

static int
parse (const char *str)
{
  size_t no = 0;
  size_t pcount = 0;
  int status;
  char buf[BUFSIZ];
  mu_address_t address = NULL;

  mu_set_user_email_domain ("localhost");
  status = mu_address_create (&address, str);
  mu_address_get_count (address, &pcount);

  if (status)
    {
      printf ("%s=> error %s\n\n", str, mu_errname (status));
      return 0;
    }
  else
    {
```

```
      printf ("%s=> pcount %lu\n", str, (unsigned long) pcount);
    }

  for (no = 1; no <= pcount; no++)
    {
      size_t got = 0;
      int isgroup;

      mu_address_is_group (address, no, &isgroup);
      printf ("%lu ", (unsigned long) no);

      if (isgroup)
        {
          mu_address_get_personal (address, no, buf, sizeof (buf), &got);
          printf ("group <%s>\n", buf);
        }
      else
        {
          mu_address_get_email (address, no, buf, sizeof (buf), 0);
          printf ("email <%s>\n", buf);
        }

      mu_address_get_personal (address, no, buf, sizeof (buf), &got);
      if (got && !isgroup)
        printf ("   personal <%s>\n", buf);

      mu_address_get_comments (address, no, buf, sizeof (buf), &got);
      if (got)
        printf ("   comments <%s>\n", buf);

      mu_address_get_local_part (address, no, buf, sizeof (buf), &got);
      if (got)
        {
          printf ("   local-part <%s>", buf);

          mu_address_get_domain (address, no, buf, sizeof (buf), &got);
          if (got)
            printf (" domain <%s>", buf);

          printf ("\n");
        }

      mu_address_get_route (address, no, buf, sizeof (buf), &got);
      if (got)
        printf ("   route <%s>\n", buf);
    }
  mu_address_destroy (&address);

  printf ("\n");
  return 0;
}

static int
parseinput (void)
{
  char buf[BUFSIZ];

  while (fgets (buf, sizeof (buf), stdin) != 0)
```

```
    {
      buf[strlen (buf) - 1] = 0;
      parse (buf);
    }

  return 0;
}

int
main (int argc, const char *argv[])
{
  argc = 1;

  if (!argv[argc])
    return parseinput ();

  for (; argv[argc]; argc++)
    {
      if (strcmp (argv[argc], "-") == 0)
parseinput ();
      else
parse (argv[argc]);
    }

  return 0;
}
```

3.1.13 Locker

```
/* Prefix mu_locker_ is reserved. */
#include <mailutils/locker.h>
```

`int mu_locker_set_default_flags` (*int* **`flags`**, *enum mu_locker_set_mode* **`mode`**) [Function]

`void mu_locker_set_default_retry_timeout` (*time_t* **`to`**) [Function]

`void mu_locker_set_default_retry_count` (*size_t* **`n`**) [Function]

`void mu_locker_set_default_expire_timeout` (*time_t* **`t`**) [Function]

`void mu_locker_set_default_external_program` (*char* ***`path`**) [Function]

A flags of 0 means that the default will be used.

`int mu_locker_create` (*mu_locker_t* *, *const char* ***`filename`**, *int* **`flags`**) [Function]

`void mu_locker_destroy` (*mu_locker_t* *) [Function]

Time is measured in seconds.

`int mu_locker_set_flags` (*mu_locker_t*, *int*) [Function]

`int mu_locker_set_expire_time` (*mu_locker_t*, *int*) [Function]

`int mu_locker_set_retries` (*mu_locker_t*, *int*) [Function]

`int mu_locker_set_retry_sleep` (*mu_locker_t*, *int*) [Function]

`int mu_locker_set_external` (*mu_locker_t*, *const char* ***`program`**) [Function]

`int mu_locker_get_flags` (*mu_locker_t, int **) [Function]

`int mu_locker_get_expire_time` (*mu_locker_t, int**) [Function]

`int mu_locker_get_retries` (*mu_locker_t, int **) [Function]

`int mu_locker_get_retry_sleep` (*mu_locker_t, int **) [Function]

`int mu_locker_get_external` (*mu_locker_t, char ***) [Function]

`int mu_locker_lock` (*mu_locker_t*) [Function]

`int mu_locker_touchlock` (*mu_locker_t*) [Function]

`int mu_locker_unlock` (*mu_locker_t*) [Function]

`int mu_locker_remove_lock` (*mu_locker_t*) [Function]

3.1.14 URL

A mailbox or a mailer can be described in a URL, the string will contain the necessary information to initialize `mailbox_t`, or `mailer_t` properly.

POP3

The POP URL scheme contains a POP server, optional port number and the authentication mechanism. The general form is

```
pop://[user[;AUTH=auth]@]host[:port]
 or
pop://[user[:passwd]@]host[:port]
```

If *:port* is omitted the default value is 110. Different forms of authentication can be specified with *;AUTH=type*. The special string *;AUTH=** indicates that the client will use a default scheme base on the capability of the server.

```
pop://obelix@gaulois.org
pop://asterix;AUTH=*@france.com
pop://falbala;AUTH=+APOP@france.com
pop://obelix;AUTH=+APOP@village.gaulois.org:2000
pop://obelix:menhir@village.gaulois.org:2000
```

For more complete information see *RFC 2368*.

IMAP

The IMAP URL scheme contains an IMAP server, optional port number and the authentication mechanism. The general form is

```
imap://[user[;AUTH=type]]@host[:port][/mailbox]
 or
imap://[user[:passwd]]@host[:port][/mailbox]
```

If *:port* is omitted the default value is 143. Different forms of authentication can be specified with *;AUTH=type*. The special string *;AUTH=** indicates that the client will use a default scheme base on the capability of the server.

```
imap://obelix@imap.gaulois.org
imap://asterix;AUTH=*@imap.france.com
imap://asterix:potion@imap.france.com
```

For more complete information see *RFC 2192*.

File

Local folder should be handle by this URL. It is preferable to let the mailbox recognize the type of mailbox and take the appropriate action.

```
file://path
file://var/mail/user
file://home/obelix/Mail
```

For MMDF, MH local mailboxes URLs are provided, but it is preferable to use `file://path` and let the library figure out which one.

```
mmdf://path
mh://path
```

Mailto

After setting a mailer, `mailto:` is used to tell the mailer where and to whom the message is for.

```
mailto://hostname
```

Mailto can be used to generate short messages, for example to subscribe to mailing lists.

```
mailto://bug-mailutils@gnu.org?body=subscribe
mailto://bug-mailutils@gnu.org?Subject=hello&body=subscribe
```

For more complete information see *RFC 2368*.

URL Functions

Helper functions are provided to retrieve and set the *URL* fields.

`int mu_url_create` (*mu_url_t* ***url**, *const char* ***name**) [Function]
Create the url data structure, but do not parse it.

`void mu_url_destroy` (*mu_url_t* ***url**) [Function]
Destroy the url and free its resources.

`int mu_url_parse` (*mu_url_t*) [Function]
Parses the url, after calling this the get functions can be called.

The syntax, condensed from *RFC 1738*, and extended with the ;auth= of *RFC 2384* (for POP) and *RFC 2192* (for IMAP) is:

```
url =
    scheme ":" [ "//"

    [ user [ ( ":" password ) | ( ";auth=" auth ) ] "@" ]

    host [ ":" port ]

    [ ( "/" urlpath ) | ( "?" query ) ] ]
```

This is a generalized URL syntax, and may not be exactly appropriate for any particular scheme.

`int mu_url_get_scheme` (*const mu_url_t, char *, size_t, size_t **) [Function]

`int mu_url_get_user` (*const mu_url_t, char *, size_t, size_t **) [Function]

`int mu_url_get_passwd` (*const mu_url_t, char *, size_t, size_t **) [Function]

`int mu_url_get_auth` (*const mu_url_t, char *, size_t, size_t **) [Function]

`int mu_url_get_host` (*const mu_url_t, char *, size_t, size_t **) [Function]

`int mu_url_get_port` (*const mu_url_t, long **) [Function]

`int mu_url_get_path` (*const mu_url_t, char *, size_t, size_t **) [Function]

`int mu_url_get_query` (*const mu_url_t, char *, size_t, size_t **) [Function]

`const char* mu_url_to_string` (*const mu_url_t*) [Function]

`int mu_url_is_scheme` (*mu_url_t, const char **scheme*) [Function]

`int mu_url_is_same_scheme` (*mu_url_t, mu_url_t*) [Function]

`int mu_url_is_same_user` (*mu_url_t, mu_url_t*) [Function]

`int mu_url_is_same_path` (*mu_url_t, mu_url_t*) [Function]

`int mu_url_is_same_host` (*mu_url_t, mu_url_t*) [Function]

`int mu_url_is_same_port` (*mu_url_t, mu_url_t*) [Function]

`char * mu_url_decode` (*const char **string*) [Function]

Decodes an *RFC 1738* encoded string, returning the decoded string in allocated memory. If the string is not encoded, this degenerates to a `strdup()`.

`int mu_url_is_ticket` (*mu_url_t ticket, mu_url_t url*) [Function]

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <mailutils/errno.h>
#include <mailutils/url.h>

int
main ()
{
  char str[1024];
  char buffer[1024];
  long port = 0;
  int len = sizeof (buffer);
  mu_url_t u = NULL;

  while (fgets (str, sizeof (str), stdin) != NULL)
    {
      int rc;

      str[strlen (str) - 1] = '\0';     /* chop newline */
      if (strspn (str, " \t") == strlen (str))
        continue;               /* skip empty lines */
```

```
      if ((rc = mu_url_create (&u, str)) != 0)
        {
          fprintf (stderr, "mu_url_create %s ERROR: [%d] %s",
                   str, rc, mu_strerror (rc));
          exit (1);
        }
      if ((rc = mu_url_parse (u)) != 0)
        {
          printf ("%s => FAILED: [%d] %s\n",
                  str, rc, mu_strerror (rc));
          continue;
        }
      printf ("%s => SUCCESS\n", str);

      mu_url_get_scheme (u, buffer, len, NULL);
      printf ("\tscheme <%s>\n", buffer);

      mu_url_get_user (u, buffer, len, NULL);
      printf ("\tuser <%s>\n", buffer);

      mu_url_get_passwd (u, buffer, len, NULL);
      printf ("\tpasswd <%s>\n", buffer);

      mu_url_get_auth (u, buffer, len, NULL);
      printf ("\tauth <%s>\n", buffer);

      mu_url_get_host (u, buffer, len, NULL);
      printf ("\thost <%s>\n", buffer);

      mu_url_get_port (u, &port);
      printf ("\tport %ld\n", port);

      mu_url_get_path (u, buffer, len, NULL);
      printf ("\tpath <%s>\n", buffer);

      mu_url_get_query (u, buffer, len, NULL);
      printf ("\tquery <%s>\n", buffer);

      mu_url_destroy (&u);

    }
  return 0;
}
```

3.1.15 Parse822

```
/* Prefix mu_parse822_ is reserved. */
#include <mailutils/parse822.h>
```

`int mu_parse822_is_char` (*char* `c`) [Function]

`int mu_parse822_is_digit` (*char* `c`) [Function]

`int mu_parse822_is_ctl` (*char* `c`) [Function]

`int mu_parse822_is_space` (*char* `c`) [Function]

`int mu_parse822_is_htab` (*char* `c`) [Function]

`int mu_parse822_is_lwsp_char (char c)` [Function]

`int mu_parse822_is_special (char c)` [Function]

`int mu_parse822_is_atom_char (char c)` [Function]

`int mu_parse822_is_q_text (char c)` [Function]

`int mu_parse822_is_d_text (char c)` [Function]

`int mu_parse822_is_smtp_q (char c)` [Function]

`int mu_parse822_skip_crlf (const char **p, const char *e)` [Function]

`int mu_parse822_skip_lwsp_char (const char **p, const char *e)` [Function]

`int mu_parse822_skip_lwsp (const char **p, const char *e)` [Function]

`int mu_parse822_skip_comments (const char **p, const char *e)` [Function]

`int mu_parse822_skip_nl (const char **p, const char *e)` [Function]

`int mu_parse822_digits (const char **p, const char *e, int min, int max, int *digits)` [Function]

`int mu_parse822_special (const char **p, const char *e, char c)` [Function]

`int mu_parse822_comment (const char **p, const char *e, char **comment)` [Function]

`int mu_parse822_atom (const char **p, const char *e, char **atom)` [Function]

`int mu_parse822_quoted_pair (const char **p, const char *e, char **qpair)` [Function]

`int mu_parse822_quoted_string (const char **p, const char *e, char **qstr)` [Function]

`int mu_parse822_word (const char **p, const char *e, char **word)` [Function]

`int mu_parse822_phrase (const char **p, const char *e, char **phrase)` [Function]

`int mu_parse822_d_text (const char **p, const char *e, char **dtext)` [Function]

`int mu_parse822_address_list (mu_address_t *a, const char *s)` [Function]

`int mu_parse822_mail_box (const char **p, const char *e, mu_address_t *a)` [Function]

`int mu_parse822_group (const char **p, const char *e, mu_address_t *a)` [Function]

`int mu_parse822_address (const char **p, const char *e, mu_address_t *a)` [Function]

`int mu_parse822_route_addr (const char **p, const char *e, mu_address_t *a)` [Function]

`int mu_parse822_route (const char **p, const char *e, char **route)` [Function]

`int mu_parse822_addr_spec` (*const char **p, const char *e, mu_address_t *a*) [Function]

`int mu_parse822_unix_mbox` (*const char **p, const char *e, mu_address_t *a*) [Function]

`int mu_parse822_local_part` (*const char **p, const char *e, char **local_part*) [Function]

`int mu_parse822_domain` (*const char **p, const char *e, char **domain*) [Function]

`int mu_parse822_sub_domain` (*const char **p, const char *e, char **sub_domain*) [Function]

`int mu_parse822_domain_ref` (*const char **p, const char *e, char **domain_ref*) [Function]

`int mu_parse822_domain_literal` (*const char **p, const char *e, char **domain_literal*) [Function]

`int mu_parse822_quote_string` (*char **quoted, const char *raw*) [Function]

`int mu_parse822_quote_local_part` (*char **quoted, const char *raw*) [Function]

`int mu_parse822_field_body` (*const char **p, const char *e, char **fieldbody*) [Function]

`int mu_parse822_field_name` (*const char **p, const char *e, char **fieldname*) [Function]

`int mu_parse822_day` (*const char **p, const char *e, int *day*) [Function]

`int mu_parse822_date` (*const char **p, const char *e, int *day, int *mon, int *year*) [Function]

`int mu_parse822_time` (*const char **p, const char *e, int *h, int *m, int *s, int *tz, const char **tz_name*) [Function]

`int mu_parse822_date_time` (*const char **p, const char *e, struct tm *tm, mu_timezone *tz*) [Function]

3.1.16 Mailcap

```
/* Prefix mu_mailcap_ is reserved. */
#include <mailutils/mailcap.h>
```

The standard *RFC 1524* (A User Agent Configuration Mechanism) suggests a file format to be used to inform a mail user agent about facilities for handling mail in various format. The configuration file is known also as mailcap and it is tipically found in UNIX platforms, a example of '`/etc/mailcap`':

```
application/pgp; gpg < %s | metamail; needsterminal; \
       test=test %{encapsulation}=entity ; copiousoutput
```

A mailcap file consits of a set of mailcap entries per line, lines beginning with '`#`' are considered comments and ignored. Long mailcap entry may be continued on multiple lines

if each line ends with a backslash character '\', the multiline will be considered a single mailcap entry. The overall format in BNF:

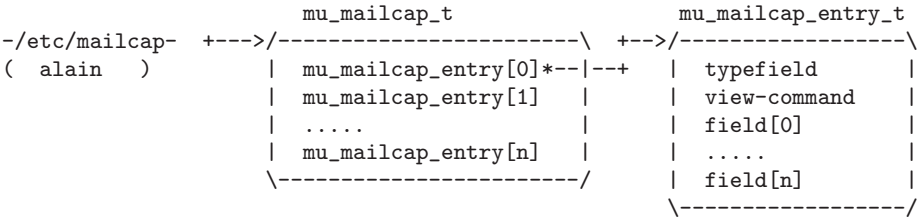
```
Mailcap-File = *mailcap-line
Mailcap-Line = comment | mailcap-entry
Comment = newline | "#" * char newline
Newline = <newline as defined by OS convention>
```

Each mailcap entry consists of a number of fields, separated by semi-colons. The first two filds are required and must occur in the secified order, the remaining fields are optional.

```
Mailcap-Entry = typefield ";" view-command ";" *[ ";" field ]
```

`mu_mailcap_t`, *mu_mailcap_entry_t* [Data Type]

The `mu_mailcap_t` and `mu_mailcap_entry_t` objects are used to hold information and it is an opaque data structure to the user. Functions are provided to retrieve information from the data structure.



An Example of Parsing a Mailcap File:

```
#include <stdlib.h>
#include <stdio.h>
#include <mailutils/mailcap.h>
#include <mailutils/stream.h>
#include <mailutils/error.h>

int
main (int argc, char **argv)
{
  mu_stream_t stream = NULL;
  int status = 0;
  char *file = argc == 1 ? "/etc/mailcap" : argv[1];
  mu_mailcap_t mailcap = NULL;

  status = mu_file_stream_create (&stream, file, MU_STREAM_READ);
  if (status)
    {
      mu_error ("cannot create file stream %s: %s",
file, mu_strerror (status));
      exit (1);
    }

  status = mu_stream_open (stream);
  if (status)
    {
      mu_error ("cannot open file stream %s: %s",
file, mu_strerror (status));
      exit (1);
    }
```

```
  status = mu_mailcap_create (&mailcap, stream);
  if (status == 0)
    {
      int i;
      size_t count = 0;
      char buffer[256];

      mu_mailcap_entries_count (mailcap, &count);
      for (i = 1; i <= count; i++)
{
  size_t j;
  mu_mailcap_entry_t entry = NULL;
  size_t fields_count = 0;

  printf ("entry[%d]\n", i);

  mu_mailcap_get_entry (mailcap, i, &entry);

  /* typefield.  */
  mu_mailcap_entry_get_typefield (entry, buffer,
  sizeof (buffer), NULL);
  printf ("\ttypefield: %s\n", buffer);

  /* view-command.  */
  mu_mailcap_entry_get_viewcommand (entry, buffer,
    sizeof (buffer), NULL);
  printf ("\tview-command: %s\n", buffer);

  /* fields.  */
  mu_mailcap_entry_fields_count (entry, &fields_count);
  for (j = 1; j <= fields_count; j++)
    {
      int status = mu_mailcap_entry_get_field (entry, j, buffer,
       sizeof (buffer), NULL);
      if (status)
{
  mu_error ("cannot retrieve field %lu: %s",
    (unsigned long) j,
    mu_strerror (status));
  break;
}
      printf ("\tfields[%d]: %s\n", j, buffer);
    }
  printf ("\n");
}
      mu_mailcap_destroy (&mailcap);
    }

  return 0;
}
```

`int mu_mailcap_create` (*mu_mailcap_t* ***mailcap***, *mu_stream_t* ***stream***) [Function]

The function allocates, parses the buffer from the *stream* and initializes *mailcap*. The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
: *mailcap* is `NULL` or *stream* is invalid.

`void mu_mailcap_destroy (`*mu_mailcap_t* ***mailcap**`)` [Function]
Release any resources from the mailcap object.

`int mu_mailcap_entries_count (`*mu_mailcap_t* **mailcap**, *size_t* ***count**`)` [Function]
The function returns the number of entries found in the mailcap. The return value is `0` on success and a code number on error conditions:

`EINVAL` *mailcap* or *count* is `NULL`.

`int mu_mailcap_get_entry (`*mu_mailcap_t* **mailcap**, *size_t* **no**, *mu_mailcap_entry_t* ***entry**`)` [Function]
Returns in *entry* the mailcap entry of *no*.

`int mu_mailcap_entry_fields_count (`*mu_mailcap_entry_t* **entry**, *size_t* ***count**`)` [Function]
The function returns the number of fields found in the entry. The return value is `0` on success and a code number on error conditions:

`EINVAL` *entry* or *count* is `NULL`.

`int mu_mailcap_entry_get_typefield (`*mu_mailcap_entry_t* **entry**, *char* ***buffer**, *size_t* **buflen**, *size_t* ***n**`)` [Function]

`int mu_mailcap_entry_get_viewcommand (`*mu_mailcap_entry_t* **entry**, *char* ***buffer**, *size_t* **buflen**, *size_t* ***n**`)` [Function]

`int mu_mailcap_entry_get_field (`*mu_mailcap_entry_t* **entry**, *size_t* **no**, *char* ***buffer**, *size_t* **buflen**, *size_t* ***n**`)` [Function]

`int mu_mailcap_entry_get_value (`*mu_mailcap_entry_t* **entry**, *const char* ***key**, *char* ***buffer**, *size_t* **buflen**, *size_t* ***n**`)` [Function]

`int mu_mailcap_entry_get_compose (`*mu_mailcap_entry_t* **entry**, *char* ***buffer**, *size_t* **buflen**, *size_t* ***n**`)` [Function]
Helper function saving in buffer, the argument of "compose" field.

`int mu_mailcap_entry_get_composetyped (`*mu_mailcap_entry_t* **entry**, *char* ***buffer**, *size_t* **buflen**, *size_t* ***n**`)` [Function]
Helper function saving in buffer, the argument of "composetyped" field.

`int mu_mailcap_entry_get_edit (`*mu_mailcap_entry_t* **entry**, *char* ***buffer**, *size_t* **buflen**, *size_t* ***n**`)` [Function]
Helper function saving in buffer, the argument of "edit" field.

`int mu_mailcap_entry_get_textualnewlines (`*mu_mailcap_entry_t* **entry**, *char* ***buffer**, *size_t* **buflen**, *size_t* ***n**`)` [Function]
Helper function saving in buffer, the argument of "textualnewlines" field.

`int mu_mailcap_entry_get_test (`*mu_mailcap_entry_t* **entry**, *char* ***buffer**, *size_t* **buflen**, *size_t* ***n**`)` [Function]
Helper function saving in buffer, the argument of "test" field.

`int mu_mailcap_entry_get_x11bitmap` (*mu_mailcap_entry_t* ***entry***, *char* ***buffer***, *size_t* ***buflen***, *size_t* ***n***) [Function]
Helper function saving in buffer, the argument of "x11-bitmap" field.

`int mu_mailcap_entry_get_description` (*mu_mailcap_entry_t* ***entry***, *char* ***buffer***, *size_t* ***buflen***, *size_t* ***n***) [Function]
Helper function saving in buffer, the argument of "description" field.

`int mu_mailcap_entry_get_nametemplate` (*mu_mailcap_entry_t* ***entry***, *char* ***buffer***, *size_t* ***buflen***, *size_t* ***n***) [Function]
Helper function saving in buffer, the argument of "nametemplate" field.

`int mu_mailcap_entry_get_notes` (*mu_mailcap_entry_t* ***entry***, *char* ***buffer***, *size_t* ***buflen***, *size_t* ***n***) [Function]
Helper function saving in buffer, the argument of "notes" field.

`int mu_mailcap_entry_needsterminal` (*mu_mailcap_entry_t* ***entry***, *int* ***on***) [Function]
Helper function. Returns **on* != 0 if the flag `needsterminal` is in the record.

`int mu_mailcap_entry_copiousoutput` (*mu_mailcap_entry_t* ***entry***, *int* ***on***) [Function]
Helper function. Returns **on* != 0 if the flag *copiousoutput* is in the record.

3.2 Authentication Library

The functions from '`libmailbox`' library get user information from the system user database. The library '`libmuauth`' extends this functionality, allowing '`libmailbox`' functions to obtain information about a user from several places, like SQL database, etc. The method used is described in detail in Section 2.2 [authentication], page 7. This chapter contains a very succinct description of the underlying library mechanism.

3.2.1 Data Types

`mu_auth_fp` [Data Type]
This is a pointer to authentication or authorization data. It is defined as follows:

```
typedef int (*mu_auth_fp) (struct mu_auth_data **return_data,
                           void *key,
                           void *func_data,
                           void *call_data);
```

Its arguments are:

return_data
Upon successful return authorization handler leaves in this memory location a pointer to the filled `mu_auth_data` structure with the user's information.

For authentication handlers this argument is always `NULL` and should be ignored.

key The search key value. Its actual type depends upon type of the handler.

For authorization handlers it is `const char*` if the handler is called by `mu_get_auth_by_name()` and `uid_t *` if it is called by `mu_get_auth_by_uid()`.

For authentication handlers it is always `struct mu_auth_data*` representing the user's data obtained by a previous call to a `mu_get_auth_by_...` function.

func_data Any data associated with this handler.

call_data Any call specific data. This argument is not used at the moment.

`mu_auth_data` [Data Type]

The `mu_auth_data` is used to return the information about the user. It is similar to system `struct passwd`, except that it is more mailutils-specific. Its definition is:

```
struct mu_auth_data {
  /* These are from struct passwd */
  char    *name;       /* user name */
  char    *passwd;     /* user password */
  uid_t   uid;         /* user id */
  gid_t   gid;         /* group id */
  char    *gecos;      /* real name */
  char    *dir;        /* home directory */
  char    *shell;      /* shell program */
  /* */
  char    *mailbox;    /* Path to the user's system mailbox */
  int     change_uid;  /* Should the uid be changed? */
};
```

`mu_auth_module` [Data Type]

The `mu_auth_module` structure contains full information about a libmuauth module. It is declared as follows:

```
struct mu_auth_module {
  char           *name;              /* Module name */
  struct argp    *argp;              /* Corresponding argp structure */
  mu_auth_fp     authenticate;       /* Authentication function ... */
  void           *authenticate_data; /* ... and its specific data */
  mu_auth_fp     auth_by_name;       /* Get user info by user name */
  void           *auth_by_name_data; /* ... and its specific data */
  mu_auth_fp     auth_by_uid;        /* Get user info by user id */
  void           *auth_by_uid_data;  /* ... and its specific data */
};
```

3.2.2 Initializing 'libmuauth'

`void mu_auth_init` (*void*) [Function]

This function registers the command line capability "auth". It must be called after registering 'libmuauth' modules and before calling `mu_agrp_parse()`. If an error occurs, this function prints diagnostic message and aborts the program.

`void MU_AUTH_REGISTER_ALL_MODULES` (*void*) [Function]

This macro registers all default modules and calls `mu_auth_init()`.

3.2.3 Module Creation and Destruction

`int mu_auth_data_alloc` (*struct mu_auth_data **`ptr`, *const char *`name`, *const char *`passwd`, *uid_t `uid`, *gid_t `gid`, *const char *`gecos`, *const char *`dir`, *const char *`shell`, *const char *`mailbox`, *int `change_uid`*) [Function]

Create a `mu_auth_data` structure and initialize it with the given values. Returns 0 on success and 1 otherwise.

`void mu_auth_data_free` (*struct mu_auth_data *`ptr`*) [Function]

Free the `mu_auth_data` structure allocated by a call to `mu_auth_data_alloc()`.

`void mu_auth_register_module` (*struct mu_auth_module *`mod`*) [Function]

Register the module defined by the *mod* argument.

3.2.4 Obtaining Authorization Information

`int mu_auth_runlist` (*list_t `flist`, struct mu_auth_data **`return_data`, *void *`key`, *void *`call_data`*); [Function]

The list is expected to contain `mu_auth_fp` pointers. Each of them is dereferenced and executed until either the list is exhausted or any of the functions returns non-zero, whichever occurs first. The *return_data* and *key* arguments are passed as the first two parameters to the function (see the definition of `mu_auth_fp`, notice the footnote), the `call_data` is passed as its last parameter.

The function returns 0 if none of the functions from `list` succeeded, i.e. returned non-zero value. Otherwise it returns the return code from the succeeded function.

`struct mu_auth_data * mu_get_auth_by_name` (*const char *`username`*) [Function]

Search the information about given user by its username. Similar to system's `getpwnam` call).

`struct mu_auth_data * mu_get_auth_by_uid` (*uid_t `uid`*) [Function]

Search the information about given user by its uid. Similar to system's `getpwuid` call).

`int mu_authenticate` (*struct mu_auth_data *`auth_data`, *char *`pass`*) [Function]

Authenticate the user whose data are in *auth_data* using password *pass*. Return 0 if the user is authenticated.

3.2.5 Existing Modules

`int mu_auth_nosupport` (*struct mu_auth_data **`return_data`, *void *`key`, *void *`func_data`, *void *`call_data`*); [Function]

The "not-supported" module. Always returns `ENOSYS`.

`mu_auth_system_module` [Variable]

This module is always registered even if '`libmuauth`' is not linked. It performs usual authentication using system user database ('`/etc/password`' et al.)

`mu_auth_generic_module` [Variable]
: This module is always registered even if '`libmuauth`' is not linked. Both its authorization handlers are `mu_auth_nosupport`. Its authentication handler computes the MD5 or DES hash over the supplied password with the seed taken from `passwd` member of its *key* argument. Then it compares the obtained hash with the `passwd` member itself and returns 1 if both strings match.

`mu_auth_pam_module` [Variable]
: Implements PAM authentication. Both authorization handlers are `mu_auth_nosupport()`.

`mu_auth_sql_module` [Variable]
: Implements authentication and authorization via MySQL database. The credentials for accessing the database are taken from global variables `sql_host`, `sql_port`, `sql_user`, `sql_passwd` and `sql_db`. The SQL queries for retrieving user information from global variables `sql_getpwnam_query` and `sql_getpwuid_query`. The variable `sql_getpass_query` keeps the query used for retrieving user's password. See Section 2.1.6 [auth], page 5, for information on command line options used to set these variables.

`mu_auth_virtual_module` [Variable]
: Implements `mu_get_auth_by_name` method using virtual mail domains. Neither `mu_get_auth_by_uid` nor `mu_authenticate` is implemented. This module must be used together with `generic` module.

3.2.6 Using '`libmuauth`' in Your Programs

To link your program against '`libmuauth`', obtain loader arguments by running `mailutils-config` as follows:

```
mailutils-config --link auth
```

See Section 2.17 [mailutils-config], page 67, for more information about this utility.

Here is a sample Makefile fragment:

```
MU_LDFLAGS=‘mailutils-config --link auth‘
MU_INCLUDES=‘mailutils-config --include‘

myprog: myprog.c
        $(CC) -omyprog $(CFLAGS) $(MU_INCLUDES) myprog.c $(MU_LDFLAGS)
```

If your program will be using only default modules provided by the library, then it will suffice to call `MU_AUTH_REGISTER_ALL_MODULES()` somewhere near the start of your program. As an example, consider the following code fragment (it is taken from the `imap4d` daemon):

```
int
main (int argc, char **argv)
{
  struct group *gr;
  int status = EXIT_SUCCESS;

  state = STATE_NONAUTH; /* Starting state in non-auth.  */

  MU_AUTH_REGISTER_ALL_MODULES ();
  mu_argp_parse (&argp, &argc, &argv, 0, imap4d_capa,
                 NULL, &daemon_param);
  ...
```

Otherwise, if your program will use it's own modules, first register them with `mu_auth_register_module` and then call `mu_auth_init()`, e.g.:

```
struct mu_auth_module radius_module = {
  ...
};

struct mu_auth_module ext_module = {
  ...
};

int
main (int argc, char **argv)
{
  mu_auth_register_module (&radius_module);
  mu_auth_register_module (&ext_module);
  mu_auth_init ();

  ...
```

These two approaches may be combined, allowing you to use both your modules and the ones provided by Mailutils. Consider the example below:

```
int
main (int argc, char **argv)
{
  mu_auth_register_module (&radius_module);
  mu_auth_register_module (&ext_module);
  MU_AUTH_REGISTER_ALL_MODULES ();

  ...
}
```

3.3 Mailutils to Scheme Interface

The library '`libmu_scm`' provides an interface between Mailutils and Guile, allowing to access the Mailutils functionality from a Scheme program. For more information about Guile, refer to section "Overview" in *The Guile Reference Manual*. For information about Scheme programming language, See section "Top" in *Revised(4) Report on the Algorithmic Language Scheme*.

3.3.1 Address Functions

`mu-address-get-personal` *address num* [Scheme procedure]
Return personal part of the *num*th email address from *address*.

`mu-address-get-comments` *address num* [Scheme procedure]
Return comment part of the *num*th email address from *address*.

`mu-address-get-email` *address num* [Scheme procedure]
Return email part of the *num*th email address from *address*.

`mu-address-get-domain` *address num* [Scheme procedure]
Return domain part of the *num*th email address from *address*.

`mu-address-get-local` *address num* [Scheme procedure]
Return local part of the *num*th email address from *address*.

`mu-address-get-count` *address* [Scheme procedure]
Return number of parts in email address *address*.

`mu-username->email` *name* [Scheme procedure]
Deduce user's email address from his username. If *name* is omitted, current username is assumed

3.3.2 Mailbox Functions

`mu-mail-directory` *url* [Scheme procedure]
If *url* is given, sets it as a name of the user's mail directory. Returns the current value of the mail directory.

`mu-folder-directory` *url* [Scheme procedure]
If *url* is given, sets it as a name of the user's folder directory. Returns the current value of the folder directory.

`mu-mailbox-open` *url mode* [Scheme procedure]
Opens the mailbox specified by *url*. *mode* is a string, consisting of the characters described below, giving the access mode for the mailbox

mode	**Meaning**
r	Open for reading.
w	Open for writing.
a	Open for appending to the end of the mailbox.
c	Create the mailbox if it does not exist.

`mu-mailbox-close` *mbox* [Scheme procedure]
Closes mailbox *mbox*.

`mu-mailbox-get-url` *mbox* [Scheme procedure]
Returns url of the mailbox *mbox*.

`mu-mailbox-get-port` *mbox mode* [Scheme procedure]
Returns a port associated with the contents of the *mbox*. *mode* is a string defining operation mode of the stream. It may contain any of the two characters: '`r`' for reading, '`w`' for writing.

`mu-mailbox-get-message` *mbox msgno* [Scheme procedure]
Retrieve from message #*msgno* from the mailbox *mbox*.

`mu-mailbox-messages-count` *mbox* [Scheme procedure]
Returns number of messages in the mailbox *mbox*.

`mu-mailbox-expunge` *mbox* [Scheme procedure]
Expunges deleted messages from the mailbox *mbox*.

`mu-mailbox-append-message` *mbox mesg* [Scheme procedure]
Appends message *mesg* to the mailbox *mbox*.

3.3.3 Message Functions

`mu-message-create` [Scheme procedure]
Creates an empty message.

`mu-message-copy` *mesg* [Scheme procedure]
Creates the copy of the message *mesg*.

`mu-message-destroy` *mesg* [Scheme procedure]
Destroys the message *mesg*.

`mu-message-set-header` *mesg header value replace* [Scheme procedure]
Sets new *value* to the header *header* of the message *mesg*. If *header* is already present in the message its value is replaced with the suplied one iff the optional *replace* is #t. Otherwise, a new header is created and appended.

`mu-message-get-size` *mesg* [Scheme procedure]
Returns the size of the message *mesg* .

`mu-message-get-lines` *mesg* [Scheme procedure]
Returns number of lines in the given message.

`mu-message-get-sender` *mesg* [Scheme procedure]
Returns email address of the sender of the message *mesg*.

`mu-message-get-header` *mesg header* [Scheme procedure]
Returns value of the header *header* from the message *mesg*.

`mu-message-get-header-fields` *mesg headers* [Scheme procedure]
Returns the list of headers in the message *mesg*. Optional argument *headers* gives a list of header names to restrict return value to.

`mu-message-set-header-fields` *mesg list replace* [Scheme procedure]
Set the headers in the message *mesg* from *list list* is a list of conses (cons HEADER VALUE). The function sets these headers in the message *mesg*. Optional parameter *replace* specifies whether the new header values should replace the headers already present in the message.

`mu-message-delete` *mesg flag* [Scheme procedure]
Mark the message *mesg* as deleted. Optional argument *flag* allows to toggle deletion mark. The message is deleted if it is `#t` and undeleted if it is `#f`

`mu-message-get-flag` *mesg flag* [Scheme procedure]
Return value of the attribute *flag* of the message *mesg*.

`mu-message-set-flag` *mesg flag value* [Scheme procedure]
Set the attribute *flag* of the message *mesg*. If optional *value* is #f, the attribute is unset.

`mu-message-get-user-flag` *mesg flag* [Scheme procedure]
Return the value of the user attribute *flag* from the message *mesg*.

`mu-message-set-user-flag` *mesg flag value* [Scheme procedure]
Set the given user attribute *flag* in the message *mesg*. If optional argumen *value* is '`#f`', the attribute is unset.

`mu-message-get-port` *mesg mode full* [Scheme procedure]
Returns a port associated with the given *mesg*. *mode* is a string defining operation mode of the stream. It may contain any of the two characters: '`r`' for reading, '`w`' for writing. If optional argument *full* is specified, it should be a boolean value. If it is '`#t`' then the returned port will allow access to any part of the message (including headers). If it is `#f` then the port accesses only the message body (the default).

`mu-message-get-body` *mesg* [Scheme procedure]
Returns the message body for the message *mesg*.

`mu-message-multipart?` *mesg* [Scheme procedure]
Returns `#t` if *mesg* is a multipart MIME message.

`mu-message-get-num-parts` *mesg* [Scheme procedure]
Returns number of parts in a multipart MIME message. Returns `#f` if the argument is not a multipart message.

`mu-message-get-part` *mesg part* [Scheme procedure]
Returns part *#part* from a multipart MIME message *mesg*.

`mu-message-send` *mesg mailer from to* [Scheme procedure]
Sends the message *mesg*. Optional *mailer* overrides default mailer settings in mu-mailer. Optional *from* and *to* give sender and recever addresses.

`mu-message-get-uid` *mesg* [Scheme procedure]
Returns uid of the message *mesg*

`mu-body-read-line` *body* [Scheme procedure]
Read next line from the *body*.

`mu-body-write` *body text* [Scheme procedure]
Append *text* to message *body*.

3.3.4 MIME Functions

`mu-mime-create` *flags mesg* [Scheme procedure]
Creates a new MIME object. Both arguments are optional. *flags* specifies the type of the object to create ('`0`' is a reasonable value). *mesg* gives the message to create the MIME object from.

`mu-mime-multipart?` *mime* [Scheme procedure]
Returns `#t` if *mime* is a multipart object.

`mu-mime-get-num-parts` *mime* [Scheme procedure]
Returns number of parts in the MIME object *mime*.

`mu-mime-get-part` *mime num* [Scheme procedure]
Returns *num*th part from the MIME object *mime*.

`mu-mime-add-part` *mime mesg* [Scheme procedure]
Adds *mesg* to the MIME object *mime*.

`mu-mime-get-message` *mime* [Scheme procedure]
Converts MIME object *mime* to a message.

3.3.5 Logging Functions

`mu-openlog` *ident option facility* [Scheme procedure]
Opens a connection to the system logger for Guile program. *ident*, *option* and *facility* have the same meaning as in openlog(3)

`mu-logger` *prio text* [Scheme procedure]
Distributes *text* via syslogd priority *prio*.

`mu-closelog` [Scheme procedure]
Closes the channel to the system logger opened by `mu-openlog`.

3.3.6 Other Functions

`mu-register-format` *rest* [Scheme procedure]
Registers desired mailutils formats. Any number of arguments can be given. Each argument must be one of the following strings:

Argument	Meaning
‘`mbox`’	Regular UNIX mbox format
‘`mh`’	MH mailbox format
‘`maildir`’	*Maildir* mailbox format
‘`pop`’	POP mailbox format
‘`imap`’	IMAP mailbox format
‘`sendmail`’	*sendmail* mailer format
‘`smtp`’	SMTP mailer format

If called without arguments, the function registers all available formats

`mu-strerror` *err* [Scheme procedure]
Return the error message corresponding to *err*, which must be an integer value.

3.3.7 Direct Linking

If you plan to link your program directly to ‘`libguile`’, it will probably make sense to link ‘`libmu_scm`’ directly as well. The arguments to the program loader may be obtained by running

```
mailutils-config --link guile
```

See Section 2.17 [mailutils-config], page 67, for more information about this utility.

Here is a sample Makefile fragment:

```
MU_LDFLAGS=‘mailutils-config --link guile‘
MU_INCLUDES=‘mailutils-config --include‘

myprog: myprog.c
        $(CC) -omyprog $(CFLAGS) $(MU_INCLUDES) myprog.c $(MU_LDFLAGS)
```

3.3.8 Dynamic Linking

Dynamic linking is the preferred method of using '`libmu_scm`'. It uses Guile "use-modules" mechanism. An interface module '`mailutils.scm`' is provided in order to facilitate using this method. This module is installed in the package data directory (by default it is '*prefix*`/share/mailutils`'). A sample use of this module is:

```
(set! %load-path (list "/usr/local/share/mailutils"))
(use-modules (mailutils))

# Now you may use mailutils functions:

(let ((mb (mu-mailbox-open "/var/spool/mail/gray" "r")))
...
```

Note, that you should explicitly modify the `%load-path` before calling `use-modules`, otherwise Guile will not be able to find '`mailutils.scm`'.

3.4 Sieve Library

`Libsieve` is GNU implementation of the mail filtering language Sieve. The library is built around a *Sieve Machine* — an abstract computer constructed specially to handle mail filtering tasks. This computer has two registers: program counter and numeric accumulator; a runtime stack of unlimited depth and the code segment. A set of functions is provided for creating and destroying instances of Sieve Machine, manipulating its internal data, compiling and executing a sieve program.

The following is a typical scenario of using `libsieve`:

1. Application program creates the instance of sieve machine.
2. Then `mu_sieve_compile` function is called to translate the Sieve source into an equivalent program executable by the Machine
3. A mailbox is opened and associated with the Machine
4. The Machine executes the program over the mailbox
5. When the execution of the program is finished, all messages upon which an action was executed other than `keep` are marked with the delete flag. Thus, running `mailbox_expunge` upon the mailbox finishes the job, leaving in the mailbox only those messages that were preserved by the filter.
6. Finally, the instance of Sieve Machine is destroyed and the resources allocated for it are reclaimed.

The following sections describe in detail the functions from the Sieve Library.

3.4.1 Sieve Data Types

`sieve_machine_t` [Data Type]

This is an opaque data type representing a pointer to an instance of sieve machine. The `sieve_machine_t` keeps all information necessary for compiling and executing the script.

It is created by `sieve_machine_create()` and destroyed by `sieve_machine_destroy()`. The functions for manipulating this data type are described in Section 3.4.2 [Manipulating the Sieve Machine], page 126.

`mu_sieve_data_type` [Data Type]

This enumeration keeps the possible types of sieve data. These are:

`SVT_VOID` No datatype.

`SVT_NUMBER`
: Numeric type.

`SVT_STRING`
: Character string.

`SVT_STRING_LIST`
: A `mu_list_t`. Each item in this list represents a character string.

`SVT_TAG` A sieve tag. See `mu_sieve_runtime_tag_t` below.

`SVT_IDENT`
: A character string representing an identifier.

`SVT_VALUE_LIST`
: A `mu_list_t`. Each item in this list is of `mu_sieve_value_t` type.

`SVT_POINTER`
: An opaque pointer.

`mu_sieve_value_t` [Data Type]

The `mu_sieve_value_t` keeps an instance of sieve data. It is defined as follows:

```
typedef struct
{
  mu_sieve_data_type type;        /* Type of the data */
  union {
    char *string;                 /* String value or identifier */
    size_t number;                /* Numeric value */
    mu_list_t list;               /* List value */
    mu_sieve_runtime_tag_t *tag;  /* Tag value */
    void *ptr;                    /* Pointer value */
  } v;
}
mu_sieve_value_t;
```

Depending on the value of `type` member, following members of the union `v` keep the actual value:

`SVT_VOID` Never appears.

`SVT_NUMBER`
: The numeric value is kept in `number` member.

`SVT_STRING`
: The string is kept in `string` member.

`SVT_STRING_LIST`
`SVT_VALUE_LIST`
: The list itself is pointed to by `list` member

`SVT_TAG` The tag value is pointed to by `tag` member.

`SVT_IDENT`
: The `string` member points to the identifier name.

`SVT_POINTER`
: The data are pointed to by `ptr` member.

`mu_sieve_tag_def_t` [Data Type]

This structure represents a definition of a tagged (optional) argument to a sieve action or test. It is defined as follows:

```
typedef struct
{
  char *name;                 /* Tag name */
  mu_sieve_data_type argtype; /* Type of tag argument. */
}
mu_sieve_tag_def_t;
```

The `name` member points to the tag's name *without leading colon*. The `argtype` is set to `SVT_VOID` if the tag does not take argument, or to the type of the argument otherwise.

`mu_sieve_runtime_tag_t` [Data Type]

This structure represents the tagged (optional) argument at a runtime. It is defined as:

```
struct mu_sieve_runtime_tag
{
  char *tag;                  /* Tag name */
  mu_sieve_value_t *arg;      /* Tag argument (if any) */
};
```

The `arg` member is `NULL` if the tag does not take an argument.

`mu_sieve_locus_t` [Data Type]

Objects of this type represent a location in the Sieve source file:

```
typedef struct
{
  const char *source_file;
  size_t source_line;
}
mu_sieve_locus_t;
```

`mu_sieve_handler_t` [Data Type]

This is a pointer to function handler for a sieve action or test. It is defined as follows:

```
typedef int (*mu_sieve_handler_t) (mu_sieve_machine_t mach,
                                   mu_list_t args,
                                   mu_list_t tags);
```

The arguments to the handler have the following meaning:

mach
: Sieve machine being processed.

args
: A list of required arguments to the handler

tags
: A list of optional arguments (tags).

`mu_sieve_printf_t` [Data Type]

A pointer to a diagnostic output function. It is defined as follows:

```
typedef int (*mu_sieve_printf_t) (void *data,
                                  const char *fmt, va_list ap);
```

data A pointer to application specific data. These data are passed as second argument to `mu_sieve_machine_init()`.

fmt Printf-like format string.

ap Other arguments.

`mu_sieve_parse_error_t` [Data Type]

This data type is declared as follows:

```
typedef int (*mu_sieve_parse_error_t) (void *data,
                                       const char *filename,
                                       int lineno,
                                       const char *fmt,
                                       va_list ap);
```

It is used to declare error handlers for parsing errors. The application-specific data are passed in the *data* argument. Arguments *filename* and *line* indicate the location of the error in the source text, while *fmt* and *ap* give verbose description of the error.

`mu_sieve_action_log_t` [Data Type]

A pointer to the application-specific logging function:

```
typedef void (*mu_sieve_action_log_t) (void *data,
                                       const mu_sieve_locus_t *locus,
                                       size_t msgno,
                                       mu_message_t msg,
                                       const char *action,
                                       const char *fmt,
                                       va_list ap);
```

data Application-specific data.

locus Location in the Sieve source file.

script Name of the sieve script being executed.

msgno Ordinal number of the message in mailbox, if appropriate. When execution is started using `sieve_message()`, this argument is zero.

msg The message this action is executed upon.

action The name of the action.

fmt
var These two arguments give the detailed description of the action.

`mu_sieve_relcmp_t` [Data Type]
`mu_sieve_relcmpn_t` [Data Type]

```
typedef int (*mu_sieve_relcmp_t) (int, int);
typedef int (*mu_sieve_relcmpn_t) (size_t, size_t);
```

`mu_sieve_comparator_t` [Data Type]

```
typedef int (*mu_sieve_comparator_t) (const char *, const char *);
```

A pointer to the comparator handler function. The function compares its two operands and returns 1 if they are equal, and 0 otherwise. *Notice*, that the sense of the return value is inverted in comparison with most standard libc functions like `stcmp()`, etc.

`mu_sieve_retrieve_t` [Data Type]

```
typedef int (*mu_sieve_retrieve_t) (void *item, void *data, int idx,
                                    char **pval);
```

A pointer to generic retriever function. See description of `mu_sieve_vlist_compare()` for details of its usage.

`mu_sieve_destructor_t` [Data Type]

```
typedef void (*mu_sieve_destructor_t) (void *data);
```

A pointer to destructor function. The function frees any resources associated with `data`. See the description of `mu_sieve_machine_add_destructor()` for more information.

`mu_sieve_tag_checker_t` [Data Type]

```
typedef int (*mu_sieve_tag_checker_t) (const char *name,
                                       mu_list_t tags,
                                       mu_list_t args)
```

A pointer to tag checker function. The purpose of the function is to perform compilation-time consistency test on tags. Its arguments are:

name Name of the test or action whose tags are being checked.

tags A list of `mu_sieve_runtime_tag_t` representing tags.

args A list of `mu_sieve_value_t` representing required arguments to *name*.

The function is allowed to make any changes in *tags* and *args*. It should return 0 if the syntax is correct and non-zero otherwise. It is responsible for issuing the diagnostics in the latter case. [FIXME: describe how to do that]

3.4.2 Manipulating the Sieve Machine

This subsection describes functions used to create an instance of the sieve machine, read or alter its internal fields and destroy it.

`int mu_sieve_machine_init (`*mu_sieve_machine_t* ***mach**, *void* ***data**`)` [Function]

The `mu_sieve_machine_init()` function creates an instance of a sieve machine. A pointer to the instance itself is returned in the argument *mach*. The user-specific data to be associated with the new machine are passed in *data* argument. The function returns 0 on success, non-zero error code otherwise,

`void mu_sieve_machine_destroy (`*mu_sieve_machine_t* ***pmach**`)` [Function]

This function destroys the instance of sieve machine pointed to by *mach* parameter. After execution of `mu_sieve_machine_destroy()` *pmach* contains `NULL`. The destructors registered with `mu_sieve_machine_add_destructor()` are executed in LIFO order.

`int mu_sieve_machine_add_destructor (`*mu_sieve_machine_t* **mach**, *mu_sieve_destructor_t* **destr**, *void* ***ptr**`);` [Function]

This function registers a destructor function *dest*. The purpose of the destructor is to free any resources associated with the item *ptr*. The destructor function takes a single argument — a pointer to the data being destroyed. All registered destructors are called in reverse order upon execution of `mu_sieve_machine_destroy()`. Here's a short example of the use of this function:

```
static void
free_regex (void *data)
{
  regfree ((regex_t*)data);
}

int
match_part_checker (const char *name, list_t tags, list_t args)
{
  regex_t *regex;

  /* Initialise the regex: */
  regex = mu_sieve_malloc (mach, sizeof (*regex));
  /* Make sure it will be freed when necessary */
  mu_sieve_machine_add_destructor (sieve_machine, free_regex, regex);
  .
  .
  .
}
```

`void * mu_sieve_get_data` (*mu_sieve_machine_t* **mach**) [Function]
This function returns the application-specific data associated with the instance of sieve machine. See `mu_sieve_machine_init()`.

`mu_message_t mu_sieve_get_message` (*mu_sieve_machine_t* **mach**) [Function]
This function returns the current message.

`size_t mu_sieve_get_message_num` (*mu_sieve_machine_t* **mach**) [Function]
This function returns the current message number in the mailbox. If there are no mailbox, i.e. the execution of the sieve code is started with `mu_sieve_message`, this function returns 1.

`int mu_sieve_get_debug_level` (*mu_sieve_machine_t* **mach**) [Function]
Returns the debug level set for this instance of sieve machine.

`mu_ticket_t mu_sieve_get_ticket` (*mu_sieve_machine_t* **mach**) [Function]
Returns the authentication ticket for this machine.

`mu_mailer_t mu_sieve_get_mailer` (*mu_sieve_machine_t* **mach**) [Function]
Returns the mailer.

`int mu_sieve_get_locus` (*mu_sieve_machine_t* **mach**, *mu_sieve_locus_t* ***locus**) [Function]
Returns the locus in the Sieve source file corresponding to the code pointer where the Sieve machine currently is.

`char * mu_sieve_get_daemon_email` (*mu_sieve_machine_t* **mach**) [Function]
This function returns the *daemon email* associated with this instance of sieve machine. The daemon email is an email address used in envelope from addresses of automatic reply messages. By default its local part is '`<MAILER-DAEMON>`' and the domain part is the machine name.

`void mu_sieve_set_error` (*mu_sieve_machine_t* **mach**, [Function]
mu_sieve_printf_t **error_printer**)

This function sets the error printer function for the machine. If it is not set, the default error printer will be used. It is defined as follows:

```
int
_sieve_default_error_printer (void *unused, const char *fmt,
                              va_list ap)
{
  return mu_verror (fmt, ap);
}
```

`void mu_sieve_set_parse_error` (*mu_sieve_machine_t* **mach**, [Function]
mu_sieve_parse_error_t **p**)

This function sets the parse error printer function for the machine. If it is not set, the default parse error printer will be used. It is defined as follows:

```
int
_sieve_default_parse_error (void *unused,
                            const char *filename, int lineno,
    const char *fmt, va_list ap)
{
  if (filename)
    fprintf (stderr, "%s:%d: ", filename, lineno);
  vfprintf (stderr, fmt, ap);
  fprintf (stderr, "\n");
  return 0;
}
```

`void mu_sieve_set_debug` (*mu_sieve_machine_t* **mach**, [Function]
mu_sieve_printf_t **debug**);

This function sets the debug printer function for the machine. If it is not set, the default debug printer is `NULL` which means no debugging information will be displayed.

`void mu_sieve_set_debug_level` (*mu_sieve_machine_t* **mach**, [Function]
mu_debug_t **dbg**, *int* **level**)

This function sets the debug level for the given instance of sieve machine. The *dbg* argument is the `mu_debug_t` object to be used with mailutils library, the *level* argument specifies the debugging level for the sieve library itself. It is a bitwise or of the following values:

`MU_SIEVE_DEBUG_TRACE`
: Trace the execution of the sieve script.

`MU_SIEVE_DEBUG_INSTR`
: Print the sieve machine instructions as they are executed.

`MU_SIEVE_DEBUG_DISAS`
: Dump the disassembled code of the sieve machine. Do not run it.

`MU_SIEVE_DRY_RUN`
: Do not executed the actions, only show what would have been done.

`void mu_sieve_set_logger` (*mu_sieve_machine_t* **mach**, [Function]
mu_sieve_action_log_t **logger**)

This function sets the logger function. By default the logger function is `NULL`, which means that the executed actions are not logged.

`void mu_sieve_set_ticket` (*mu_sieve_machine_t* **mach**, *mu_ticket_t* **ticket**) [Function]
This function sets the authentication ticket to be used with this machine.

`void mu_sieve_set_mailer` (*mu_sieve_machine_t* **mach**, *mu_mailer_t* **mailer**) [Function]
This function sets the mailer. The default mailer is `"sendmail:"`.

`void mu_sieve_set_daemon_email` (*mu_sieve_machine_t* **mach**, *const char* ***email**) [Function]
This functions sets the *daemon email* for `reject` and `redirect` actions.

`int mu_sieve_is_dry_run` (*mu_sieve_machine_t* **mach**) [Function]
The `mu_sieve_is_dry_run()` returns 1 if the machine is in *dry run* state, i.e. it will only log the actions that would have been executed without actually executing them. The dry run state is set by calling `mu_sieve_set_debug_level()` if its last argument has the `MU_SIEVE_DRY_RUN` bit set.

`const char * mu_sieve_type_str` (*mu_sieve_data_type* **type**) [Function]
Returns the string representation for the given sieve data type. The return value is a pointer to a static constant string.

3.4.3 Logging and Diagnostic Functions

`void mu_sieve_error` (*mu_sieve_machine_t* **mach**, *const char* ***fmt**, ...) [Function]
Format and output an error message using error printer of the machine *mach*.

`void mu_sieve_debug` (*mu_sieve_machine_t* **mach**, *const char* ***fmt**, ...) [Function]
Format and output a debug message using debug printer of the machine *mach*.

`void mu_sieve_log_action` (*mu_sieve_machine_t* **mach**, *const char* ***action**, *const char* ***fmt**, ...) [Function]
Log a sieve action using logger function associated with the machine *mach*.

`void mu_sieve_abort` (*mu_sieve_machine_t* **mach**) [Function]
Immediately abort the execution of the script.

3.4.4 Symbol Space Functions

`mu_sieve_register_t * mu_sieve_test_lookup` (*mu_sieve_machine_t* **mach**, *const char* ***name**) [Function]
Find a register object describing the test *name*. Returns `NULL` if no such test exists.

`mu_sieve_register_t * mu_sieve_action_lookup` (*mu_sieve_machine_t* **mach**, *const char* ***name**) [Function]
Find a register object describing the action *name*. Returns `NULL` if no such action exists.

`int mu_sieve_register_test` (*mu_sieve_machine_t* **`mach`**, *const char* ***`name`**, *mu_sieve_handler_t* **`handler`**, *mu_sieve_data_type* ***`arg_types`**, *mu_sieve_tag_group_t* ***`tags`**, *int* **`required`**) [Function]

`int mu_sieve_register_action` (*mu_sieve_machine_t* **`mach`**, *const char* ***`name`**, *mu_sieve_handler_t* **`handler`**, *mu_sieve_data_type* ***`arg_types`**, *mu_sieve_tag_group_t* ***`tags`**, *int* **`required`**) [Function]

`int mu_sieve_register_comparator` (*mu_sieve_machine_t* **`mach`**, *const char* ***`name`**, *int* **`required`**, *mu_sieve_comparator_t* **`is`**, *mu_sieve_comparator_t* **`contains`**, *mu_sieve_comparator_t* **`matches`**, *mu_sieve_comparator_t* **`regex`**, *mu_sieve_comparator_t* **`eq`**) [Function]

`int mu_sieve_tag_lookup` (*mu_list_t* **`taglist`**, *char* ***`name`**, *mu_sieve_value_t* ****`arg`**) [Function]

`int mu_sieve_load_ext` (*mu_sieve_machine_t* **`mach`**, *const char* ***`name`**) [Function]

3.4.5 Memory Allocation

The following functions act as their libc counterparts. The allocated memory is associated with the *mach* argument and is automatically freed upon the call to `mu_sieve_machine_destroy (mach)`.

`void * mu_sieve_malloc` (*mu_sieve_machine_t* **`mach`**, *size_t* **`size`**) [Function]
Allocates *size* bytes and returns a pointer to the allocated memory.

`char * mu_sieve_mstrdup` (*mu_sieve_machine_t* **`mach`**, *const char* ***`str`**) [Function]
This function returns a pointer to a new string which is a duplicate of the string *str*.

`void * mu_sieve_mrealloc` (*mu_sieve_machine_t* **`mach`**, *void* ***`ptr`**, *size_t* **`size`**) [Function]
Changes the size of the memory block pointed to by *ptr* to *size* bytes. The contents will be unchanged to the minimum of the old and new sizes; newly allocated memory will be uninitialized. If *ptr* is `NULL`, the call is equivalent to `mu_sieve_malloc(mach, size)`; if *size* is equal to zero, the call is equivalent to `mu_sieve_mfree(ptr)`. Unless *ptr* is `NULL`, it must have been returned by an earlier call to `mu_sieve_malloc()` or `mu_sieve_mrealloc()`.

`void mu_sieve_mfree` (*mu_sieve_machine_t* **`mach`**, *void* ***`ptr`**) [Function]
`mu_sieve_mfree()` frees the memory space pointed to by *ptr* and detaches it from the destructor list of *mach*. The *ptr* must have been returned by a previous call to `mu_sieve_malloc()` or `mu_sieve_mrealloc()`. Otherwise, or if `mu_sieve_mfree(ptr)` has already been called before, undefined behaviour occurs.

If *ptr* is `NULL`, no operation is performed.

3.4.6 Compiling and Executing the Script

`int mu_sieve_compile` (*mu_sieve_machine_t* **`mach`**, *const char* ***`name`**) [Function]
Compile the sieve script from the file *name*.

`int mu_sieve_mailbox` (*mu_sieve_machine_t* **mach**, *mu_mailbox_t* **mbox**) [Function]
Execute the code from the given instance of sieve machine *mach* over each message in the mailbox *mbox*.

`int mu_sieve_message` (*mu_sieve_machine_t* **mach**, *mu_message_t* **message**) [Function]
Execute the code from the given instance of sieve machine *mach* over the *message*.

`int mu_sieve_disass` (*mu_sieve_machine_t* **mach**) [Function]
Dump the disassembled code of the sieve machine *mach*.

3.4.7 Writing Loadable Commands

This section contains an example of how to write external loadable commands for GNU libsieve.

```
/*  This is an example on how to write extension tests for GNU sieve.
    It provides test "numaddr".

    Syntax:   numaddr [":over" / ":under"] <header-names: string-list>
              <limit: number>

    The "numaddr" test counts Internet addresses in structured headers
    that contain addresses.  It returns true if the total number of
    addresses satisfies the requested relation:

    If the argument is ":over" and the number of addresses is greater than
    the number provided, the test is true; otherwise, it is false.

    If the argument is ":under" and the number of addresses is less than
    the number provided, the test is true; otherwise, it is false.

    If the argument is empty, ":over" is assumed. */

#ifdef HAVE_CONFIG_H
# include <config.h>
#endif

#include <stdlib.h>
#include <mailutils/libsieve.h>

struct val_ctr {  /* Data passed to the counter function */
  mu_header_t hdr;   /* Headers of the current message */
  size_t limit;   /* Limit for the number of addresses */
  size_t count;   /* Number of addresses counted so far */
};

/* Count addresses in a single header value.

   Input:
     ITEM is the name of the header to scan.
     DATA is a pointer to the val_ctr structure
   Return value:
     non-zero if the limit on the number of addresses has been reached. */

static int
_count_items (void *item, void *data)
```

```
{
  char *name = item;
  struct val_ctr *vp = data;
  char *val;
  mu_address_t addr;
  size_t count = 0;

  if (mu_header_aget_value (vp->hdr, name, &val))
    return 0;
  if (mu_address_create (&addr, val) == 0)
    {
      mu_address_get_count (addr, &count);
      mu_address_destroy (&addr);
      vp->count += count;
    }
  free (val);
  return vp->count >= vp->limit;
}

/* Handler for the numaddr test */
static int
numaddr_test (mu_sieve_machine_t mach, mu_list_t args, mu_list_t tags)
{
  mu_sieve_value_t *h, *v;
  struct val_ctr vc;
  int rc;

  if (mu_sieve_get_debug_level (mach) & MU_SIEVE_DEBUG_TRACE)
    {
      mu_sieve_locus_t locus;
      mu_sieve_get_locus (mach, &locus);
      mu_sieve_debug (mach, "%s:%lu: NUMADDR\n",
   locus.source_file,
   (unsigned long) locus.source_line);
    }

  /* Retrieve required arguments: */
  /* First argument: list of header names */
  h = mu_sieve_value_get (args, 0);
  if (!h)
    {
      mu_sieve_error (mach, "numaddr: can't get argument 1");
      mu_sieve_abort (mach);
    }
  /* Second argument: Limit on the number of addresses */
  v = mu_sieve_value_get (args, 1);
  if (!v)
    {
      mu_sieve_error (mach, "numaddr: can't get argument 2");
      mu_sieve_abort (mach);
    }

  /* Fill in the val_ctr structure */
  mu_message_get_header (mu_sieve_get_message (mach), &vc.hdr);
  vc.count = 0;
  vc.limit = v->v.number;

  /* Count the addresses */
```

```
  rc = mu_sieve_vlist_do (h, _count_items, &vc);

  /* Here rc >= 1 iff the counted number of addresses is greater or equal
     to vc.limit. If ':under' tag was given we reverse the return value */
  if (mu_sieve_tag_lookup (tags, "under", NULL))
    rc = !rc;
  return rc;
}

/* Syntactic definitions for the numaddr test */

/* Required arguments: */
static mu_sieve_data_type numaddr_req_args[] = {
  SVT_STRING_LIST,
  SVT_NUMBER,
  SVT_VOID
};

/* Tagged arguments: */
static mu_sieve_tag_def_t numaddr_tags[] = {
  { "over", SVT_VOID },
  { "under", SVT_VOID },
  { NULL }
};

static mu_sieve_tag_group_t numaddr_tag_groups[] = {
  { numaddr_tags, NULL },
  { NULL }
};

/* Initialization function. It is the only function exported from this
   module. */
int
SIEVE_EXPORT(numaddr,init) (mu_sieve_machine_t mach)
{
  return mu_sieve_register_test (mach, "numaddr", numaddr_test,
                                 numaddr_req_args, numaddr_tag_groups, 1);
}
```

4 Sieve Language

The input language understood by the GNU Sieve Library is a superset of the Sieve language as described in RFC 3028.

4.1 Lexical Structure

Whitespace and Comments

Comments are semantically equivalent to whitespace and can be used anyplace that whitespace is (with one exception in multi-line strings, as described below).

There are two kinds of comments: hash comments, that begin with a '`#`' character that is not contained within a string and continue until the next newline, and C-style or bracketed comments, that are delimited by '`/*`' and '`*/`' tokens. The bracketed comments may span multiple lines. E.g.:

```
if size :over 100K
  { # this is a comment
    discard;
  }

if size :over 100K
  { /* this is a comment
       this is still a comment */ discard /* this is a comment again
     */ ;
  }
```

Like in C, bracketed comments do not nest.

Lexical Tokens

The basic lexical entities are *identifiers* and *literals*.

An *identifier* is a sequence of letters, digits and underscores, started with a letter or underscore. For example, `header` and `check_822_again` are valid identifiers, whereas `1st` is not. A special form of identifier is *tag*: it is an identifier prefixed with a colon ('`:`'), e.g.: `:comparator`.

A *literal* is a data that is not executed, merely evaluated "as is", to be used as arguments to commands. There are four kinds of literals:

- Number

  Numbers are given as ordinary unsigned decimal numbers. An optional suffix may be used to indicate a multiple of a power of two. The suffixes are: '`K`' specifying "kibi-", or 1,024 (2^10) times the value of the number; '`M`' specifying "mebi-", or 1,048,576 (2^20) times the value of the number; and '`G`' specifying "tebi-", or 1,073,741,824 (2^30) times the value of the number.

  The numbers have 32 bits of magnitude.

- String

  A *string* is any sequence of characters enclosed in double quotes ('`"`'). A string cannot contain newlines and double quote characters. This limitation will disappear in future releases.

- Multiline Strings

  A *multiline string* is used to represent large blocks of text with embedded newlines and special characters. It starts with the keyword `text:` followed by a newline and ends with a dot (‘`.`’) on a newline by itself. Any characters between these two markers are taken verbatim. For example:

```
text:
** This is an authomatic response from my message **
** filtering program.                              **

I can not attend your message right now. However it
will be saved, and I will read it as soon as I am back.

Regards,
Fred
.
```

  Notice that a hashed comment or whitespace may occur between `text:` and the newline. However, when used inside the multiline string a hash sign looses its special meaning (except in one case, see below) and is taken as is, as well as bracketed comment delimiters. In other words, no comments are allowed within a multiline string. E.g.:

```
text: # This is a comment

Sample text
# This line is taken verbatim
/* And this line too */
.
```

  The only exception to this rule is that preprocessor `include` statement is expanded as usual when found within a multiline string (see Section 4.3 [Preprocessor], page 139), e.g.:

```
text:
#include <myresponse.txt>
.
```

  This results in the contents of file ‘`myresponse.txt`’ being read and interpreted as the contents of the multiline string.

  GNU libsieve extends the described syntax as follows. If the keyword `text:` is immediately followed by a dash (‘`-`’), then all leading tab characters are stripped from input lines and the line containing delimiter (‘`.`’). This allows multiline strings within scripts to be indented in a natural fashion.

  Furthermore, if the `text:` (optionally followed by ‘`-`’) is immediately followed by a word, this word will be used as ending delimiter of multiline string instead of the default dot. For example:

```
if header "from" "me@example.com"
  {
    reject text:-EOT
        I do not accept messages from
        this address.
        .
        .
        EOT
     # Notice that this the multiline string ends here.
     # The single dots above will be part of it.
    ;
  }
```

- String Lists

  A *string list* is a comma-delimited list of quoted strings, enclosed in a pair of square brackets, e.g.:

  ```
  ["me@example.com", "me00@landru.example.edu"]
  ```

  For convenience, in any context where a list of strings is appropriate, a single string is allowed without being a member of a list: it is equivalent to a list with a single member. For example, the following two statements are equivalent:

  ```
  exists "To";
  exists ["To"];
  ```

4.2 Syntax

Being designed for the sole purpose of filtering mail, Sieve has a very simple syntax.

4.2.1 Commands

The basic syntax element is a *command*. It is defined as follows:

```
command-name [tags] args
```

where *command-name* is an identifier representing the name of the command, *tags* is an optional list of *optional* or *tagged arguments* and *args* is a list of *required* or *positional arguments*.

Positional arguments are literals delimited with whitespace. They provide the command with the information necessary to its proper functioning. Each command has a fixed number of positional arguments. It is an error to supply more arguments to the command or to give it fewer arguments than it accepts.

Optional arguments allow to modify the behaviour of the command, like command line options in UNIX do. They are a list of *tags* (see Section 4.1 [Lexical Structure], page 135) separated by whitespace. An optional argument may have at most one parameter.

Each command understands a set of optional arguments. Supplying it tags that it does not understand results in an error.

For example, consider the following command

```
header :mime :comparator "i;octet" ["to", "from"] "bug-mailutils@gnu.org"
```

Here, given that `header` takes two positional arguments: `header` is command name, the list `["to", "from"]` is first positional argument and the string `"bug-mailutils@gnu.org"` is second positional argument. There are two optional arguments: `:mime` and `:comparator`. The latter has a string `"i;octet"` as its parameter.

4.2.2 Actions Described

An *action* is a Sieve command that performs some operation over the message. Actions do the main job in any Sieve program. Syntactically, an action is a command terminated with semicolon, e.g.:

```
keep;

fileinto "mbox";
```

GNU Sieve provides the full set of actions described in RFC 3028. It also allows to extend this set using loadable actions. See Section 4.7 [Actions], page 146, for detailed discussion of actions.

4.2.3 Control Flow

The only control flow statement Sieve has is "if" statement. In its simplest form it is:

```
if condition { ... }
```

The effect of this statement is that the sequence of actions between the curly braces is executed only if the `condition` evaluates to `true`.

A more elaborate form of this statement allows to execute two different sets of actions depending on whether the condition is true or not:

```
if condition { ... } else { ... }
```

The most advanced form of the "if" statement allows to select an action depending on what condition from the set of conditions is met.

```
if cond1 { ... } elsif cond2 { ... } else { ... }
```

There may be any number of "elsif" branches in an "if" statement. However it may have at most one "else" branch. Notes for C programmers:

1. The braces surrounding each branch of an "if" statement are required.
2. The "else if" construct is disallowed. Use "elsif" keyword instead.

Here's an example of "if" statement:

```
if header :contains "from" "coyote"
  {
    discard;
  }
elsif header :contains ["subject"] ["$$$"]
  {
    discard;
  }
else
  {
    fileinto "INBOX";
  }
```

The following section describes in detail conditions used in "if" statements.

4.2.4 Tests and Conditions

Tests are Sieve commands that return boolean value. E.g. the test

```
header :contains "from" "coyote"
```

returns true only if the header "From" of the current message contains substring "coyote".

The tests shipped with the GNU Sieve are described in Section 4.6 [Tests], page 141.

Condition is a Sieve expression that evaluates to `true` or `false`. In its simplest form, condition is just a Sieve test.

To reverse the sense of a condition use keyword `not`, e.g.:

```
not header :contains "from" "coyote"
```

The results of several conditions may be joined together by logical `and` and `or` operations. The special form `allof` takes several tests as its arguments and computes the logical `and` of their results. Similarly, the form `anyof` performs logical `or` over the results of its arguments. E.g.:

```
if anyof (not exists ["From", "Date"],
          header :contains "from" "fool@example.edu")
  {
    discard;
  }
```

4.3 Preprocessor

The preprocessor statements are a GNU extension to the Sieve language. The syntax for a preprocessor statement is similar to that used in `C` programming language, i.e.: a pound character ('`#`') followed by a preprocessor directive and its arguments. Any amount of white-space can be inserted between the '`#`' and the directive. Currently implemented directives are `include` and `searchpath`.

Sieve #include directive

The `#include` directive reads in the contents of the given file. The contents is "inserted" into the text being parsed starting at the line where the directive appears. The directive takes two forms:

`#include "`*`filename`*`"`
: The *filename* is taken relative to the current directory.

`#include <`*`filename`*`>"`
: The *filename* is searched in the list of include directories as specified by the '`-I`' command line options.

If *filename* starts with a directory separator character ('`/`') both forms have the same effect.

Sieve #searchpath directive

The `#searchpath` directive adds its argument to the list of directories searched for loadable modules. It has the same effect as '`-L`' command line switch used by GNU sieve utility (see Section 2.1.9 [sieve group], page 7).

4.4 Require Statement

```
Syntax:   require string;
          require string-list;
```

The require statement informs the parser that a script makes use of a certain extension. Multiple capabilities can be declared using the second form of the statement. The actual handling of a capability name depends on its suffix.

If the name starts with '`comparator-`', it is understood as a request to use the specified comparator. The comparator name consists of the characters following the suffix.

If the name starts with '`test-`', it means a request to use the given test. The test name consists of the characters following the suffix.

Otherwise, the capability is understood as a name of an action to be used.

The `require` statement, if present, must be used before any other statement that is using the required capability. As an extension, the GNU sieve allows the `require` and any other statements to be interspersed.

By default the following actions and comparators are always required:

- stop
- keep
- discard
- i;octet
- i;ascii-casemap

Example:

```
require ["fileinto", "reject"];

require "fileinto";

require "comparator-i;ascii-numeric";
```

When processing arguments for `require` statement, GNU libsieve uses the following algorithm:

1. Look up the name in a symbol table. If the name begins with '`comparator-`' it is looked up in the comparator table. If it begins with '`test-`', the test table is used instead. Otherwise the name is looked up in the action table.
2. If the name is found, the search is terminated.
3. Otherwise, transform the name. First, any '`comparator-`' or '`test-`' prefix is stripped. Then, any character other than alphanumeric characters, '`.`' and '`,`' is replaced with dash ('`-`'). The name thus obtained is used as a file name of an external loadable module.
4. Try to load the module. The module is searched in the following search paths (in the order given):
   1. Mailutils module directory. By default it is '`$prefix/lib/mailutils`'.
   2. Sieve library path as given with the '`-L`' options in the command line
   3. Additional search directories specified with the `#searchpath` directive.
   4. The value of the environment variable `LTDL_LIBRARY_PATH`.
   5. System library search path: The system dependent library search path (e.g. on Linux it is set by the contents of the file '`/etc/ld.so.conf`' and the value of the environment variable `LD_LIBRARY_PATH`).

   The value of `LTDL_LIBRARY_PATH` and `LD_LIBRARY_PATH` must be a colon-separated list of absolute directories, for example, '`"/usr/lib/mypkg:/lib/foo"`'.

   In any of these directories, `libsieve` first attempts to find and load the given filename. If this fails, it tries to append the following suffixes to the file name:

1. the libtool archive extension '`.la`'
2. the extension used for native dynamic libraries on the host platform, e.g., '`.so`', '`.sl`', etc.

5. If the module is found, `libsieve` executes its initialization function (see below) and again looks up the name in the symbol table. If found, search terminates successfully.
6. If either the module is not found, or the symbol wasn't found after execution of the module initialization function, search is terminated with an error status. `libsieve` then issues the following diagnostic message:

```
source for the required action NAME is not available
```

4.5 Comparators

GNU libsieve supports the following built-in comparators:

`i;octet` This comparator simply compares the two arguments octet by octet

`i;ascii-casemap`
: It treats uppercase and lowercase characters in the ASCII subset of UTF-8 as the same. This is the default comparator.

`i;ascii-numeric`
: Treats the two arguments as ASCII representation of decimal numbers and compares their numeric values. This comparator must be explicitly required prior to use.

4.6 Tests

This section describes the built-in tests supported by GNU libsieve. In the discussion below the following macro-notations are used:

match-type
: This tag specifies the matching type to be used with the test. It can be one of the following:

  `:is` The `:is` match type describes an absolute match; if the contents of the first string are absolutely the same as the contents of the second string, they match. Only the string "frobnitzm" is the string "frobnitzm". The null key ":is" and only ":is" the null value. This is the default match-type.

  `:contains`
  : The `:contains` match type describes a substring match. If the value argument contains the key argument as a substring, the match is true. For instance, the string "frobnitzm" contains "frob" and "nit", but not "fbm". The null key "" is contained in all values.

  `:matches` The `:matches` version specifies a wildcard match using the characters '`*`' and '`?`'. '`*`' matches zero or more characters, and '`?`' matches a single character. '`?`' and '`*`' may be escaped as '`\\?`' and '`\\*`' in strings to match against themselves. The first backslash escapes the second backslash; together, they escape the '`*`'.

`:regex` The `:regex` version specifies a match using POSIX Extended Regular Expressions.

`:value` *relation*
The `:value` match type does a relational comparison between strings. Valid values for *relation* are:

"eq"	Equal
"ne"	Not Equal
"gt"	Greater Than
"ge"	Greater than or Equal
"lt"	Less Than
"le"	Less than or Equal

`:count` *relation*
This match type first determines the number of the specified entities (headers, addresses, etc.) in the message and does a relational comparison of the number of entities to the values specified in the test expression. The test expression must be a list of one element.

comparator
A *comparator* syntax item is defined as follows:

```
:comparator "comparator-name"
```

It instructs sieve to use the given comparator with the test. If *comparator-name* is not one of '`i;octet`', '`i;ascii-casemap`' it must be required prior to using it. For example:

```
require "comparator-i;ascii-numeric";

if header :comparator "i;ascii-numeric" :is "X-Num" "10"
  {
    ...
```

address-part
This syntax item is used when testing structured Internet addresses. It specifies which part of an address must be used in comparisons. Exactly one of the following tags may be used:

`:all` Use the whole address. This is the default.

`:localpart`
Use local part of the address.

`:domain` Use domain part of the address.

Notice, that *match-type* modifiers interact with comparators. Some comparators are not suitable for matching with `:contains` or `:matches`. If this occurs, sieve issues an appropriate error message. For example, the statement:

```
if header :matches :comparator "i;ascii-numeric"
```

would result in the following error message:

```
comparator ‘i;ascii-numeric’ is incompatible with match type ‘:matches’
in call to ‘header’
```

`false` [Test]

This test always evaluates to "false".

`true` [Test]

This test always evaluates to "true".

`address` [*address-part*][*comparator*][*match-type*] *header-names* *key-list* [Test]

Tagged arguments:

address-part
: Selects the address part to compare. Default is the whole email address (`:all`).

comparator
: Specifies the comparator to be used instead of the default `i;ascii-casemap`.

match-type
: Specifies the match type to be used instead of the default `:is`.

Required arguments:

header-names
: A list of header names.

key-list
: A list of address values.

The `address` test matches Internet addresses in structured headers that contain addresses. It returns `true` if any header contains any key in the specified part of the address, as modified by *comparator* and *match-type* optional arguments.

This test returns `true` if any combination of the *header-names* and *key-list* arguments match.

The `address` primitive never acts on the phrase part of an email address, nor on comments within that address. Use the `header` test instead. It also never acts on group names, although it does act on the addresses within the group construct.

Example:

```
if address :is :all "from" "tim@example.com"
  {
     discard;
  }
```

`size` [*:over*|*:under*] *number* [Test]

The `size` test deals with the size of a message. The required argument *number* represents the size of the message in bytes. It may be suffixed with the following quantifiers:

'`k`'
'`K`'
: The number is expressed in kilobytes.

'`m`'
'`M`'
: The number is expressed in megabytes.

‘g’
‘G’ The number is expressed in gigabytes.

If the tagged argument is ‘:over’, and the size of the message is greater than *number*, the test is true; otherwise, it is false.

If the argument is ‘:under’, and the size of the message is less than the *number*, the test is true; otherwise, it is false.

Otherwise, the test is true only if the size of the message equals exactly *number*. This is a GNU extension.

The size of a message is defined to be the number of octets from the initial header until the last character in the message body.

`envelope` [*address-part*][*comparator*][*match-type*] *envelope-part key-list* [Test]

Tagged arguments:

address-part
Selects the address part to compare. Default is the whole email address (`:all`).

comparator
Specifies the comparator to be used instead of the default `i;ascii-casemap`.

match-type
Specifies the match type to be used instead of the default `:is`.

Required arguments:

envelope-parts
A list of envelope parts to operate upon.

key-list A list of address values.

The `envelope` test is true if the specified part of the SMTP envelope matches the specified key.

If the envelope-part strings is (case insensitive) ‘`from`’, then matching occurs against the FROM address used in the `SMTP MAIL` command.

Notice, that due to the limitations imposed by SMTP envelope structure the use of any other values in *envelope-parts* header is meaningless.

`exists` *header-names* [Test]

Required arguments:

header-names
List of message header names.

The `exists` test is `true` if the headers listed in *header-names* argument exist within the message. All of the headers must exist or the test is false.

The following example throws out mail that doesn't have a From header and a Date header:

```
if not exists ["From","Date"]
  {
     discard;
  }
```

`header` [*comparator*] [*match-type*] [*:mime*] *header-names key-list* [Test]

Tagged arguments:

comparator
: Specifies the comparator to be used instead of the default `i;ascii-casemap`.

match-type
: Specifies the match type to be used instead of the default `:is`.

:mime
: This tag instructs `header` to search through the mime headers in multipart messages as well.

Required arguments:

header-names
: A list of header names.

key-list
: A list of header values.

The `header` test evaluates to true if any header name matches any key. The type of match is specified by the optional match argument, which defaults to ":is" if not explicitly given.

The test returns `true` if any combination of the *header-names* and *key-list* arguments match.

If a header listed in *header-names* exists, it contains the null key (‘""’). However, if the named header is not present, it does not contain the null key. So if a message contained the header

```
X-Caffeine: C8H10N4O2
```

these tests on that header evaluate as follows:

```
header :is ["X-Caffeine"] [""] ⇒ false
header :contains ["X-Caffeine"] [""] ⇒ true
```

`numaddr` [*:over*|*:under*] *header-names number* [Test]

This test is provided as an example of loadable extension tests. You must use ‘`require "test-numaddr"`’ statement before actually using it.

The `numaddr` test counts Internet addresses in structured headers that contain addresses. It returns true if the total number of addresses satisfies the requested relation.

If the tagged argument is ‘`:over`’ and the number of addresses is greater than *number*, the test is true; otherwise, it is false.

If the tagged argument is ‘`:under`’ and the number of addresses is less than *number*, the test is true; otherwise, it is false.

If the tagged argument is not given, ‘`:over`’ is assumed.

4.7 Actions

The GNU libsieve supports the following default actions:

- stop
- keep
- discard
- fileinto
- reject
- redirect

Among them the first three actions do not need to be explicitly required by a `require` statement, while the others do.

These actions are described in detail below.

`stop` [Action]

The `stop` action ends all processing. If no actions have been executed, then the `keep` action is taken.

`keep` [Action]

The effect of this action is to preserve the current message in the mailbox. This action is executed if no other action has been executed.

`discard` [Action]

`Discard` silently throws away the current message. No notification is returned to the sender, the message is deleted from the mailbox.

Example:

```
if header :contains ["from"] ["idiot@example.edu"]
  {
    discard;
  }
```

`fileinto` *folder* [Action]

Required arguments:

folder A string representing the folder name

The `fileinto` action delivers the message into the specified folder.

`reject` *reason* [Action]

The optional `reject` action refuses delivery of a message by sending back a message delivery notification to the sender. It resends the message to the sender, wrapping it in a "reject" form, noting that it was rejected by the recipient. The required argument *reason* is a string specifying the reason for rejecting the message.

Example:

If the message contained

```
Date: Tue, 1 Apr 1997 09:06:31 -0800 (PST)
From: coyote@desert.example.org
To: roadrunner@acme.example.com
Subject: I have a present for you
```

```
I've got some great birdseed over here at my place.
Want to buy it?
```

and the user's script contained:

```
if header :contains "from" "coyote@desert.example.org"
  {
    reject "I am not taking mail from you, and I don't want
            your birdseed, either!";
  }
```

then the original sender <coyote@desert.example.org> would receive the following notification:

```
To: <coyote@desert.example.org>
X-Authentication-Warning: roadrunner set sender using -f flag
Content-Type: multipart/mixed; boundary===== =_aaaaaaaaaa0
MIME-Version: 1.0
----- =_aaaaaaaaaa0
The original message was received at
Tue, 1 Apr 1997 09:07:15 -0800 from
coyote@desert.example.org.
Message was refused by recipient's mail filtering program.
Reason given was as follows:

I am not taking mail from you, and I don't want your birdseed, either!

----- =_aaaaaaaaaa0
Content-Type: message/delivery-status

Reporting-UA: sieve; GNU Mailutils 0.1.3
Arrival-Date: Tue, 1 Apr 1997 09:07:15 -0800
Final-Recipient: RFC822; roadrunner@acme.example.com
Action: deleted
Disposition: automatic-action/MDN-sent-automatically;deleted
Last-Attempt-Date: Tue, 1 Apr 1997 09:07:15 -0800

----- =_aaaaaaaaaa0
Content-Type: message/rfc822

From: coyote@desert.example.org
To: roadrunner@acme.example.com
Subject: I have a present for you

I've got some great birdseed over here at my place.
Want to buy it?
----- =_aaaaaaaaaa0
```

If the *reason* argument is rather long, the common approach is to use the combination of the `text:` and `#include` keywords, e.g.:

```
if header :mime :matches "Content-Type"
          [ "*application/msword;*", "*audio/x-midi*" ]
  {
    reject text:
#include "nomsword.txt"
    .
    ;
  }
```

`redirect` *address* [Action]

The `redirect` action is used to send the message to another user at a supplied *address*, as a mail forwarding feature does. This action makes no changes to the message body or existing headers, but it may add new headers. It also modifies the envelope recipient.

The `redirect` command performs an MTA-style "forward" — that is, what you get from a '`.forward`' file using `sendmail` under UNIX. The address on the SMTP envelope is replaced with the one on the `redirect` command and the message is sent back out. *Notice*, that it differs from the MUA-style forward, which creates a new message with a different sender and message ID, wrapping the old message in a new one.

4.8 GNU Extensions

This section summarizes the GNU extensions to the sieve language

1. Multiline strings syntax

   GNU libsieve understands the following multiline string syntax:

   ```
   text:[-][delimiter]
   ....
   delimiter
   ```

   The meaning of optional flags is the same as in shell "here document" construct: the dash strips all leading tab characters from the string body, thus allowing it to be indented in a natural fashion; *delimiter* introduces the new end-of-text delimiter instead of the default dot. If *delimiter* starts with a backslash, no preprocessing will be performed within a string.

2. Handling of the `require` statement.
   - According to the RFC an error must occur if a `require` appears after a command other than `require`. The GNU sieve library allows interspersing the `require` and other statements. The only requirement is that `require` must occur before a statement that is using the required capability (see Section 4.4 [Require Statement], page 139).
   - Prefixing the required capability with "test" requires the use of an extension test.

3. `header` test

   The `header` takes an optional argument `:mime`, meaning to scan the headers from each part of a multipart message.

4. `size` test

   The `size` test allows to omit the optional argument (:over|:under). In this case exact equality is assumed.

5. `envelope` test

   The only value that can be meaningfully used as the first required argument of an `envelope` test is '`from`'. This limitation may disappear from the subsequent releases.

6. Match type optional argument.

   Along with the usual `:is`, `:matches` and `contains` matching type, GNU sieve library understands `:regex` type. This matching type toggles POSIX Extended Regular Expression matching.

5 Reporting Bugs

Email bug reports to bug-mailutils@gnu.org. Be sure to include the word "mailutils" somewhere in the "Subject:" field.

As the purpose of bug reporting is to improve software, please be sure to include maximum information when reporting a bug. The information needed is:

- Version of the package you are using.
- Compilation options used when configuring the package.
- Conditions under which the bug appears.

The archives of bug-mailutils mailing list are available from http://mail.gnu.org/mailman/listinfo/bug-mailutils.

6 Getting News About GNU Mailutils

The two places to look for any news regarding GNU Mailutils are the Mailutils homepage at http://www.gnu.org/software/mailutils, and the project page at http://savannah.gnu.org/projects/mailutils.

The updated versions of this manual are available online from http://www.gnu.org/software/mailutils/manual.

7 Acknowledgement

In no particular order,

- Jakob Kaivo jkaivo@ndn.net,
- Jeff Bailey jbailey@gnu.org,
- Sean Perry shaleh@debian.org,
- Thomas Fletcher thomasf@qnx.com,
- Dave Inglis dinglis@qnx.com,
- Brian Edmond briane@qnx.com,
- Sam Roberts sroberts@uniserve.com,
- Sergey Poznyakoff gray@Mirddin.farlep.net,
- François Pinard pinard@IRO.UMontreal.CA.
- Jordi Mallach jordi@sindominio.net
- Wojciech Polak polak@gnu.org

Appendix A References

- SMTP
  - *RFC 2821: Simple Mail Transfer Protocol*
  - *RFC 2368: The mailto URL scheme*
- POP3
  - *RFC 1939: Post Office Protocol - Version 3*
  - *RFC 1734: POP3 AUTHentication command*
  - *RFC 1957: Some Observations on Implementations of the Post Office Protocol (POP3)*
  - *RFC 2449: POP3 Extension Mechanism*
  - *RFC 2384: POP URL Scheme*
- IMAP4
  - *RFC 2060: INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*
  - *RFC 2088: IMAP4 non-synchronizing literals*
  - *RFC 2193: IMAP4 Mailbox Referrals*
  - *RFC 2221: IMAP4 Login Referrals*
  - *RFC 2342: IMAP4 Namespace*
  - *RFC 2192: IMAP URL Scheme*
  - *RFC 1731: IMAP4 Authentication Mechanisms*
  - *RFC 2245: Anonymous SASL Mechanism*
- message formats
  - *RFC 2822: Internet Message Format*
  - *RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*
  - *RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*
  - *RFC 2047: Multipurpose Internet Mail Extensions (MIME) Part Three: Message Header Extensions for Non-ASCII Text*
  - *RFC 2049: Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples*
  - *RFC 2111: Content-ID and Message-ID Uniform Resource Locators*
- miscellaneous related topics
  - *RFC 1738: Uniform Resource Locators (URL)*
  - *RFC 2298: An Extensible Message Format for Message Disposition Notifications*
  - *RFC 3028: Sieve: A Mail Filtering Language*
  - *RFC 3431: Sieve Extension: Relational Tests*
  - *Internet Email Protocols: A Developer's Guide, by Kevin Johnson*

Appendix B GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and

that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called

an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

B.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Function Index

This is an alphabetical list of all Mailutils functions.

N

P

R

S

T

U

V

Variable Index

Keyword Index

R

S

T

U

V

W

Z

Program Index

Concept Index

This is a general index of all issues discussed in this manual