GNU Mailutils

version 3.19, 2 January 2025

Alain Magloire, Sergey Poznyakoff et al.

Published by the Free Software Foundation, 31 Milk Street, Boston, MA 02196, USA

Short Contents

Table of Contents

1 Introduction

GNU Mailutils is a set of libraries and utilities for handling electronic mail. It addresses a wide audience and can be of interest to application developers, casual users and system administrators alike.

It provides programmers with a consistent API allowing them to handle a variety of different mailbox formats transparently and without having to delve into complexities of their internal structure. While doing so, it also provides interfaces that simplify common programming tasks, such as handling lists, parsing configuration files, etc. The philosophy of Mailutils is to have a single and consistent programming interface for various objects designed to handle the same task. It tries to use their similarities to create an interface that hides their differences and complexities. This covers a wide variety of programming tasks: apart from mailbox handling, Mailutils also contains a unified iterface for work with various DBM databases and much more.

The utilities built upon these libraries share that same distinctive feature: no matter what is the internal structure of an object, it is always handled the same way as other objects that do the same task. Again, the most common example of this approach are, of course, mailboxes. Whatever Mailutils program you use, you can be sure it is able to handle various mailbox formats. You even don't have to inform it about what type a mailbox is: it will do its best to discover it automatically.

This approach sometimes covers entities which are seldom regarded as compatible. For example, using Mailutils it is possible to treat an SMTP connection as a mailbox opened only for appending new messages. This in turn, provides a way for extending the functionality of some utilities. As an example, using this concept of mailboxes, the usual mail delivery agent becomes able to do things usually reserved for mail transport agents only!

At the core of Mailutils is `libmailutils`, a library which provides an API for accessing a generalized mailbox. A set of complementary libraries provide methods for handling particular mailbox implementations: UNIX mailbox, Maildir, MH, POP3, IMAP4, even SMTP. Mailutils offers functions for almost any mail-related task, such as parsing of messages, email addresses and URLs, handling MIME messages, listing mail folders, mailcap facilities, extensible Sieve filtering, access control lists. It supports various modern data security and authentication techniques: TLS encryption, SASL and GSSAPI, to name a few. Mailutils is able to work with a wide variety of authorization databases, ranging from traditional system password database up to RADIUS, SQL and LDAP.

The utilities provided by Mailutils include `imap4d` and `pop3d` mail servers, mail reporting utility `comsatd`, general-purpose mail delivery agent `maidag`, mail filtering program `sieve`, an implementation of MH message handling system and much more.

All utilities share the same subset of command line options and use a unified configuration mechanism, which allows to easily configure the package as a whole.

1.1 What this Book Contains

This book addresses a wide audience of both system administrators and users that aim to use Mailutils programs, and programmers who wish to use Mailutils libraries in their programs. Given this audience, the book is divided in three major parts.

The first part provides a detailed description of each Mailutils utility, and advices on how to use them in various situations. This part is intended for users and system administrators who are using Mailutils programs. If you are not interested in programming using Mailutils, this is the only part you need to read.

Subsequent parts address programmers.

The second part is a tutorial which provides an introduction to programming techniques for writing mail applications using GNU Mailutils.

Finally, the third part contains a complete Mailutils library reference.

This version of the book is not finished. The places that may contain inaccurate information carry prominent notices stating so. For updated versions of the documentation, visit `http://mailutils.org/manual`. All material that ends up in this document is first published in the Mailutils Wiki, available at `http://mailutils.org/wiki`. Be sure to visit it for latest updates.

If you have any questions that are not answered there, feel free to ask them at the mailing list `bug-mailutils@gnu.org`.

1.2 A bit of History, and why use this package?

==========

Editor's note:

The information in this node may be obsolete or otherwise inaccurate. This message will disappear, once this node revised.

==========

This package started off to try and handle large mailbox files more gracefully then available at that time 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 Mailbox

The principal object Mailutils operates on is *mailbox* – a collection of mail messages. The two main characteristics of a mailbox are its type and path. The *type* defines how the messages are stored within a mailbox. The *path* specifies the location of the mailbox. The two characteristics are usually combined within a *Uniform Resource Locator* (URL), which uniquely identifies the mailbox. The syntax for URL is:

type`:[//[`*user*`:`*password*`@]`*host*`[:`*port*`]]`*path*`[?`*query*`][;`*params*`]`

The square brackets do not appear in a URL, instead they are used to denote optional parts.

Not all parts are meaningful for all types. Their usage and purpose are described in the sections that follow.

2.1 Local Mailboxes

Local mailboxes store mail in files on the local file system. A local mailbox URL is:

type`://`*path*`[;`*params*`]`

The *path* defines its location in the file system. For example:

```
mbox:///var/spool/mail/gray
```

Optional *params* is a semicolon-separated list of optional arguments that configures indexed directory structure. See [local URL parameters], page 19, for a detailed description.

The local mailbox types are:

mbox
: A traditional UNIX mailbox format. Messages are stored sequentially in a single file. Each message begins with a '`From`' line, identifying its sender and date when it was received. A single empty line separates two adjacent messages.

  This is the default format.

maildir
: The *Maildir* mailbox format. Each message is kept in a separate file with a unique name. Each mailbox is therefore a directory. This mailbox format eliminates file locking and makes message access much faster.

  This format was originally described by D. J. Bernstein in `http://cr.yp.to/proto/maildir.html`.

mh
: MH Message Handling System format. Each message is kept in a separate file named after its sequential numeric identifier within the mailbox. Deleted messages are not removed, but instead the corresponding file is renamed by prepending a comma to its original name. Each mailbox is a directory. Mailboxes can be nested.

  This format was originally developed by RAND Corporation. Mailutils implementation is compatible both with the original implementation and with its descendant *nmh*.

file
: This type can be used when accessing an existing mailbox of any of the formats defined above. The actual mailbox format is determined automatically. This type is assumed when a mailbox is referred to by its full pathname.

2.2 Remote Mailboxes

Remote mailboxes are accessed via one of the remote message protocols.

The basic remote mailbox types are:

pop
: Remote mailbox accessed using the *Post Office Protocol* (POP3). Default port number 110.

  The URL is:

  `pop://[`*user*`[:`*pass*`][;auth=+APOP]@]`*host*`[:`*port*`][;notls]`

  The *host* gives the name or IP address of the host running a POP3 server. Optional *port* can be used to connect to a port other than the default 110.

  The *user* and *pass* supply authentication credentials. If any of them is missing, Mailtils will first try to obtain it from the ticket file. If that fails, the behavior depends on the type of the controlling terminal. If the terminal is a tty device (i.e. the program accessing the mailbox was started from the command line), it will ask the user to supply the missing credentials. Otherwise it will issue an appropriate error message and refuse to access the mailbox.

  By default, the usual POP3 authentication is used. The '`auth=+APOP`' authentication parameter instructs Mailutils to use APOP authentication instead.

  If the server offers the STLS capability, Mailutils will attempt to establish encrypted TLS connection. The '`notls`' parameter disables this behavior.

pops
: Remote mailbox accessed using the *Post Office Protocol* (POP3). The transmission channel is encrypted using the *transport layer security* (TLS). The default port is 995.

  The URL is:

  `pops://[`*user*`[:`*pass*`][;auth=+APOP]@]`*host*`[:`*port*`]`

  The meaning of its components is the same as for '`pop`' type.

imap
: Remote mailbox accessed via the *Internet Message Access Protocol*. Default port number is 143.

  The URL is:

  `imap://[`*user*`[:`*pass*`]@]`*host*`[:`*port*`][;notls]`

  The *host* gives the name or IP address of the host running a IMAP4 server. Optional *port* can be used to connect to a port other than the default 143.

  The *user* and *pass* supply authentication credentials. If any of them is missing, Mailtils will first try to obtain it from the ticket file. If that fails, the behavior depends on the type of the controlling terminal. If the terminal is a tty device (i.e. the program accessing the mailbox was started from the command line), it will ask the user to supply the missing credentials. Otherwise it will issue an appropriate error message and refuse to access the mailbox.

  If the server offers the STARTTLS capability, Mailutils will attempt to establish encrypted TLS connection. The '`notls`' parameter disables this behavior.

imaps
: The '`imaps`' type differs in that its transmission channel is encrypted using the *transport layer security* (TLS). The default port is 993.

The URL is:

`imaps://[user[:pass]@]host[:port]`

The meaning of its components is the same as for '`imap`' type.

2.3 SMTP Mailboxes

SMTP mailboxes types are special remote mailboxes that allow only append operation. Appending a message is equivalent to sending it to the given recipient or recipients.

smtp A remote mailbox accessed using the Simple Message Transfer Protocol.

The SMTP URL syntax is:

`smtp://[user[:pass][;auth=mech,...]@]host[:port][;params]`

The *host* gives the name or IP address of the host running SMTP server. Optional *port* can be used to connect to a port other than the default 25.

The *user*, *pass*, and '`auth=`' elements supply credentials for ESMTP authentication, if the server supports it.

If the ESMTP authentication is used, Mailutils will select the best authentication mechanism from the list offered by the server. To force it to use a particular authentication mechanism, use the '`auth`' authentication parameter. Its value is a comma-separated list of authentication mechanisms, in the order from the most to the least preferred one, e.g.:

```
smtp://smith:guessme;auth=cram-md5,digest-md5@localhost
```

Optional *params* is a semicolon-separated list of additional parameters. Valid parameters are:

domain=*string*
: Append '`@string`' to those recipient addresses that lack the domain part.

from=*addr*
: Use *addr* as sender address.

noauth
: Disable ESMTP authentication.

notls
: Disable TLS.

recipient-headers[=*name*[,*name*...]]
: Use the supplied header names to determine recipient addresses. When no values are supplied, disables header scanning.

strip-domain
: Strip domain part from all recipient addresses.

to=*addr*[,*addr*...]
: Deliver messages to the supplied email addresses.

smtps A remote mailbox accessed using the Simple Message Transfer Protocol, with the transmission channel encrypted using the *transport layer security* (TLS). The default port is 465.

The URL is

```
smtps://[user[:pass][;auth=mech,...]@]host[:port][;params]
```

See the '`smtp`' type for a detailed description of its types. The only difference from '`smtp`' is that the '`notls`' parameter is not used.

2.4 Program Mailboxes

Program mailboxes support only append operation. Appending a message is performed by invoking the specified program and passing the message to its standard input stream.

A '`sendmail`' mailbox is identified by the following URL:

```
sendmail[://path]
```

The messages are sent by invoking `sendmail` binary with the `-oi -t` options. If the message being appended has the '`From:`' header, its value is passed to `sendmail` using the `-f` option.

The default path to the sendmail binary is system-dependent. The *path* part can be used to specify it explicitly.

The '`prog`' mailbox URL is:

```
prog://pathname[?query]
```

Messages are appended by invoking the program *pathname* with the arguments supplied by *query*. The latter is a list of words delimited by '`&`' characters.

Arguments can contain the following variables (see Section 3.2.2 [Variables], page 15):

sender — Expands to the sender email address.

rcpt — Expands to comma-separated list of email addresses obtained from '`To:`', '`Cc:`' and '`Bcc:`' headers of the message.

3 Mailutils Programs

GNU Mailutils provides a broad set of utilities for handling electronic mail. These utilities address the needs of both system administrators and users.

All utilities are built around a single core subsystem and share many common aspects. All of them are able to work with almost any existing mailbox formats. They use a common configuration file syntax, and their configuration files are located in a single subdirectory.

In this chapter we will discuss each utility, and give some advices on how to use them in various real life situations.

First of all we will describe command line and configuration file syntax.

3.1 Command Line

3.1.1 Basic Notions About Command Line Options

Many command line options have two forms, called short and long forms. Both forms are absolutely identical in function; they are interchangeable.

The *short* form is a traditional form for UNIX utilities. In this form, the option consists of a single dash, followed by a single letter, e.g. `-c`.

Short options which require arguments take their arguments immediately following the option letter, optionally separated by white space. For example, you might write `-f name`, or `-fname`. Here, `-f` is the option, and `name` is its argument.

Short options which allow optional arguments take their arguments immediately following the option letter, *without any intervening white space characters*. This is important, so that the command line parser might discern that the text following option is its argument, not the next command line parameter. For example, if option `-d` took an optional argument, then `-dname` would mean the option with its argument (`name` in this case), and `-d name` would mean the `-d` option without any argument, followed by command line argument `name`.

Short options' letters may be clumped together, but you are not required to do this. When short options are clumped as a set, use one (single) dash for them all, e.g. `-cvl` is equivalent to `-c -v -l`. However, only options that do not take arguments may be clustered this way. If an option takes an argument, it can only be the last option in such a cluster, otherwise it would be impossible to specify the argument for it. Anyway, it is much more readable to specify such options separated.

The *long* option names are probably easier to memorize than their short counterparts. They consist of two dashes, followed by a multi-letter option name, which is usually selected to be a mnemonics for the operation it requests. For example, `--verbose` is a long option that increases the verbosity of a utility. In addition, long option names can abbreviated, provided that such an abbreviation is unique among the options understood by a given utility. For example, if a utility takes options `--foreground` and `--forward`, then the shortest possible abbreviations for these options are `--fore` and `--forw`, correspondingly. If you try to use `--for`, the utility will abort and inform you that the abbreviation you use is ambiguous, so it is not clear which of the options you intended to use.

Long options which require arguments take those arguments following the option name. There are two ways of specifying a mandatory argument. It can be separated from the

option name either by an equal sign, or by any amount of white space characters. For example, if the `--file` option requires an argument, and you wish to supply `name` as its argument, then you can do so using any of the following notations: `--file=name` or `--file name`.

In contrast, optional arguments must always be introduced using an equal sign.

3.1.2 Options That are Common for All Utilities.

All GNU Mailutils programs understand a common subset of options.

`--help`
`-?` Display a short summary of the command line options understood by this utilities, along with a terse description of each.

The output of this option consists of three major parts. First, a usage synopsis is displayed. For example:

```
Usage: sieve [OPTION...] SCRIPT
GNU sieve -- a mail filtering tool
```

The first line tells that the `sieve` utility takes any number of options (brackets indicate optional part) and a single mandatory argument (‘`SCRIPT`’). The second lines summarizes the purpose of the utility.

Following this header is an option summary. It consists of two columns:

```
-c, --compile-only          Compile script and exit
-d, --debug[=FLAGS]         Debug flags
-e, --email=ADDRESS         Override user email address
```

The leftmost column contains a comma-separated list of option names. Short options are listed first. The options are ordered alphabetically. Arguments, if any, are specified after the last option name in the list, so that, e.g. the option ‘`-e`’ in the example above requires an argument: ‘`-e ADDRESS`’. Optional arguments are enclosed in square brackets, as in `--debug` option in the example above.

The rightmost column contains a short description of the option purpose.

The last part of `--help` output contains some additional notices and lists the email address for reporting bugs.

`--usage` Display a short summary of options. In the contrast to the `--help` option, only option names and arguments are printed, without any textual description. For example:

```
Usage: sieve [-cv?V] [--compile-only] [--debug[=FLAGS]]
             [--email=ADDRESS] SCRIPT
```

The exact formatting of the output produced by these two options is configurable. See Appendix D [Usage Vars], page 217, for a detailed descriptions of it.

`--version`
`-V` Print program version and exit.

`--show-config-options`
Show configuration options used when compiling the package. You can use this option to verify if support for a particular mailbox format or other functionality

is compiled in the binary. The output of this option is intended to be both machine-readable and understandable by humans.

The following command line options affect parsing of configuration files. Here we provide a short summary, the next section will describe them in detail.

`--config-file=`*`file`*
: Load this configuration file, instead of the default.

`--config-help`
: Show configuration file summary.

`--config-lint`
: Check configuration file syntax and exit

`--config-verbose`
: Verbosely log parsing of the configuration files.

`--no-site-config`
: Do not load site-wide configuration file.

`--no-user-config`
: Do not load user configuration file.

`--no-config`
: Don't load site-wide and user configuration files.

`--set=`*`path=value`*
: Set configuration variable. See [the –set option], page 14.

3.2 Mailutils Configuration File

Configuration files are the principal means of configuring any GNU Mailutils component. When started, each utility tries to load its configuration from the following locations, in that order:

1. Main site-wide configuration file.

   It is named *`sysconfdir`*`/mailutils.conf`, where *sysconfdir* stands for the system configuration directory set when compiling the package. You can obtain the value of *sysconfdir* by running

   ```
   $ mailutils info sysconfdir
   ```

   or

   ```
   $ prog --show-config-options | grep SYSCONFDIR
   ```

   where *prog* stands for any GNU Mailutils utility.

   The site-wide configuration file is not read if any of `--no-site-config` or `--no-config` command line options was given.

   Older versions of GNU Mailutils read configuration from file `mailutils.rc`. To facilitate transition, mailutils will look for that file as well. If both the default site-wide configuration file and legacy configuration file are present you will get the following warning:

   ```
   legacy configuration file /etc/mailutils.rc ignored
   ```

Otherwise, if `mailutils.conf` does not exist and `mailutils.rc` is present, it will be used instead and the following warning will be issued:

```
using legacy configuration file /etc/mailutils.rc:
please rename it to /etc/mailutils.conf
```

2. Per-user configuration file.

   Client utilities, such as `frm` or `sieve`, look in the user home directory for a file named '*.prog*', where *prog* is the name of the utility. If present, this file will be loaded after loading the site-wide configuration file. For example, the per-user configuration file for `sieve` utility is named `.sieve`.

   Loading of per-user configuration file is disabled by `--no-user-config` and `--no-config` options.

Server programs, such as `imap4d` don't use per-user configuration files.

The `--no-config` option provides a shortcut for disabling loading of the default configuration files. For servers, its effect is the same as of `--no-site-config`. For client utilities, it is equivalent to `--no-site-config --no-user-config` used together.

The `--config-file` command line option instructs the program to read configuration from the file supplied as its argument. In that case, default configuration files are not used at all.

Neither site-wide nor user configuration files are required to exist. If any or both of them are absent, GNU Mailutils won't complain – the utility will silently fall back to its default settings.

To make configuration processing more verbose, use the `--config-verbose` command line option. Here is an example of what you might get using this option:

```
imap4d: parsing file `/etc/mailutils.conf'
imap4d: finished parsing file `/etc/mailutils.conf'
```

Specifying this option more than once adds more verbosity to this output. If this option is given two times, GNU Mailutils will print each configuration file statement it parsed, along with the exact location where it occurred (the exact meaning of each statement will be described later in this chapter):

```
imap4d: parsing file `/etc/mailutils.conf'
# 1 "/etc/mailutils.conf"
mailbox {
# 2 "/etc/mailutils.conf"
  mailbox-pattern maildir:/var/spool/mail;type=index;param=2;user=${user};
# 3 "/etc/mailutils.conf"
  mailbox-type maildir;
};
# 6 "/etc/mailutils.conf"
include /etc/mailutils.d;
imap4d: parsing file `/etc/mailutils.d/imap4d'
...
```

To test configuration file without actually running the utility, use the `--config-lint` command line option. With this option, any Mailutils utility exits after finishing parsing of the configuration files. Any errors occurred during parsing are displayed on the standard

error output. This option can be combined with `--config-verbose` to obtain more detailed output.

The `--config-help` command line option produces on the standard output the summary of all configuration statements understood by the utility, with detailed comments and in the form suitable for configuration file. For example, the simplest way to write a configuration file for, say, `imap4d` is to run

```
$ imap4d --config-help > imap4d.conf
```

and to edit the `imap4d.conf` file with your editor of choice.

The order in which configuration files are loaded defines the precedence of their settings. Thus, for client utilities, settings from the per-user configuration file override those from the site-wide configuration.

It is also possible to set or override arbitrary configuration variables in the command line. It can be done via the `--set` option. Its argument is a *pathname* of the variable to be set, followed by an equals sign and a value. For example, to define the variable '`syslog`' in section '`logging`' to '`no`', do the following:

```
$ imap4d --set .logging.syslog=no
```

Configuration pathnames are discussed in detail in Section 3.2.1.3 [Paths], page 14. For a detailed description of this option, [the –set option], page 14.

The `--set` options are processed after loading all configuration files.

3.2.1 Configuration File Syntax

The configuration file consists of statements and comments.

There are three classes of lexical tokens: keywords, values, and separators. Blanks, tabs, newlines and comments, collectively called *white space* are ignored except as they serve to separate tokens. Some white space is required to separate otherwise adjacent keywords and values.

3.2.1.1 Comments

Comments may appear anywhere where white space may appear in the configuration file. There are two kinds of comments: single-line and multi-line comments. *Single-line* comments start with '`#`' or '`//`' and continue to the end of the line:

```
# This is a comment
// This too is a comment
```

Multi-line or *C-style* comments start with the two characters '`/*`' (slash, star) and continue until the first occurrence of '`*/`' (star, slash).

Multi-line comments cannot be nested. However, single-line comments may well appear within multi-line ones.

3.2.1.2 Statements

A *simple statement* consists of a keyword and value separated by any amount of whitespace. Simple statement is terminated with a semicolon ('`;`').

The following is a simple statement:

```
standalone yes;
```

```
pidfile /var/run/pop3d.pid;
```

A *keyword* begins with a letter and may contain letters, decimal digits, underscores ('_') and dashes ('-'). Examples of keywords are: '`expression`', '`output-file`'.

A *value* can be one of the following:

number
: A number is a sequence of decimal digits.

boolean
: A boolean value is one of the following: '`yes`', '`true`', '`t`' or '`1`', meaning *true*, and '`no`', '`false`', '`nil`', '`0`' meaning *false*.

unquoted string
: An unquoted string may contain letters, digits, and any of the following characters: '`_`', '`-`', '`.`', '`/`', '`@`', '`*`', '`:`'.

quoted string
: A quoted string is any sequence of characters enclosed in double-quotes ('`"`'). A backslash appearing within a quoted string introduces an *escape sequence*, which is replaced with a single character according to the following rules:

Sequence	Replaced with
\a	Audible bell character (ASCII 7)
\b	Backspace character (ASCII 8)
\f	Form-feed character (ASCII 12)
\n	Newline character (ASCII 10)
\r	Carriage return character (ASCII 13)
\t	Horizontal tabulation character (ASCII 9)
\v	Vertical tabulation character (ASCII 11)
\\	A single backslash ('\')
\"	A double-quote.

Table 3.1: Backslash escapes

In addition, the sequence '`\`*`newline`*' is removed from the string. This allows to split long strings over several physical lines, e.g.:

```
"a long string may be\
 split over several lines"
```

If the character following a backslash is not one of those specified above, the backslash is ignored and a warning is issued.

Two or more adjacent quoted strings are concatenated, which gives another way to split long strings over several lines to improve readability. The following fragment produces the same result as the example above:

```
"a long string may be"
" split over several lines"
```

Here-document
: A *here-document* is a special construct that allows to introduce strings of text containing embedded newlines.

The `<<word` construct instructs the parser to read all the following lines up to the line containing only *word*, with possible trailing blanks. Any lines thus read are concatenated together into a single string. For example:

```
<<EOT
A multiline
string
EOT
```

The body of a here-document is interpreted the same way as a double-quoted string, unless *word* is preceded by a backslash (e.g. '`<<\EOT`') or enclosed in double-quotes, in which case the text is read as is, without interpretation of escape sequences.

If *word* is prefixed with `-` (a dash), then all leading tab characters are stripped from input lines and the line containing *word*. Furthermore, if `-` is followed by a single space, all leading whitespace is stripped from them. This allows to indent here-documents in a natural fashion. For example:

```
<<- TEXT
    The leading whitespace will be
    ignored when reading these lines.
TEXT
```

It is important that the terminating delimiter be the only token on its line. The only exception to this rule is allowed if a here-document appears as the last element of a statement. In this case a semicolon can be placed on the same line with its terminating delimiter, as in:

```
help-text <<-EOT
        A sample help text.
EOT;
```

list

A *list* is a comma-separated list of values. Lists are enclosed in parentheses. The following example shows a statement whose value is a list of strings:

```
alias (test,null);
```

In any case where a list is appropriate, a single value is allowed without being a member of a list: it is equivalent to a list with a single member. This means that, e.g.

```
alias test;
```

is equivalent to

```
alias (test);
```

A *block statement* introduces a logical group of statements. It consists of a keyword, followed by an optional value, and a sequence of statements enclosed in curly braces, as shown in the example below:

```
server srv1 {
  host 10.0.0.1;
  community "foo";
}
```

The closing curly brace may be followed by a semicolon, although this is not required.

3.2.1.3 Statement Path

`Mailutils` configuration files have a distinct hierarchical structure. Each statement in such files can therefore be identified by its name and the names of block statements containing it. Such names form the *pathname*, similar to that used by UNIX file system.

For example, consider the following file:

```
foo {
  bar {
    baz 45;   # A.
  }
  baz 98;     # B.
}
```

The full pathname of the statement marked with '`A`' can be written as:

```
.foo.bar.baz
```

Similarly, the statement marked with '`B`' has the following pathname:

```
.foo.baz
```

The default path component separator is dot. A pathname beginning with a component separator is called *absolute pathname*. Absolute pathnames uniquely identify corresponding statements. If the leading dot is omitted, the resulting pathname is called *relative*. Relative pathnames identify statements in relation to the current point of reference in the configuration file.

Any other punctuation character can be used as a component separator, provided that it appears at the beginning of the pathname. In other words, only absolute pathnames allow for a change in component separators.

A block statement that has a tag is referred to by the statement's name, followed by an equals sign, followed by the tag value. For example, the statement '`A`' in the file below:

```
program x {
  bar {
    baz 45;   # A.
  }
}
```

is identified by the following pathname:

```
.program=x.bar.baz
```

The tag can optionally be enclosed in a pair of double quotes. Such a quoting becomes mandatory for tags that contain white space or path component separator, e.g.:

```
.program="a.out".bar.baz
```

The `--set` command line option allows you to set configuration variables from the command line. Its argument consists of the statement path and value, separated by a single equals sign (no whitespace is permitted at either side of it). For example, the following option:

```
--set .logging.facility=mail
```

has the same effect as the following statement in the configuration file:

```
logging {
```

```
        facility mail;
    }
```

Values set using this option override those set in the configuration files. This provides a convenient way for temporarily changing configuration without altering configuration files.

Notice, that when using `--set`, the '`=`' sign has two purposes: first it separates statement path from the value, thus forming an assignment, and secondly it can be used within the path itself to introduce a tag. To illustrate this, let's assume you have the following statement in your configuration file:

```
program pop3d {
    logging {
       facility mail;
    }
    server 0.0.0.0 {
       transcript no;
    }
}
```

Now assume you wish to temporarily change logging facility to '`local1`'. The following option will do this:

```
--set .program=pop3d.logging.facility=local1
```

When splitting the argument to `--set`, the option parser always looks for the rightmost equals sign. Everything to the right of it is the value, and everything to the left of it - the path.

If the tag contains dots (as the `server` statement in the example above), you should either escape them with slashes or change the pathname separator to some other character, e.g.:

```
--set .program=pop3d.server='0\.0\.0\.0'.transcript=yes
```

or

```
--set /program=pop3d/server="0.0.0.0"/transcript=yes
```

3.2.2 Configuration Variables

Certain configuration statements allow for the use of variable references in their values. A variable reference has the form '`$variable`' or '`${variable}`', where *variable* is the variable name. It is expanded to the actual value of *variable* when Mailutils consults the configuration statement in question.

The two forms are entirely equivalent. The form with curly braces is normally used if the variable name is immediately followed by an alphanumeric symbol, which will otherwise be considered part of it. This form also allows for specifying the action to take if the variable is undefined or expands to an empty value.

During variable expansion, the forms below cause Mailutils to test for a variable that is unset or null. Omitting the colon results in a test only for a variable that is unset.

${*variable*:-*word*}
: *Use Default Values.* If *variable* is unset or null, the expansion of *word* is substituted. Otherwise, the value of *variable* is substituted.

${*variable*:=*word*}
: *Assign Default Values.* If *variable* is unset or null, the expansion of *word* is assigned to variable. The value of *variable* is then substituted.

${*variable*:?*word*}
: *Display Error if Null or Unset.* If *variable* is null or unset, the expansion of *word* (or a message to that effect if *word* is not present) is output to the current logging channel. Otherwise, the value of *variable* is substituted.

${*variable*:+*word*}
: *Use Alternate Value.* If *variable* is null or unset, nothing is substituted, otherwise the expansion of *word* is substituted.

When a value is subject to variable expansion, it is also subject to *command expansion*. Commands are invoked in string values using the following format:

```
$(cmd arg)
```

where *cmd* is the command name, and *args* is a list of arguments separated by whitespace. Arguments can in turn contain variable and command references.

The following commands are defined:

`localpart` *`string`* [Command]
: Treats *string* as an email address and returns the part preceding the '`@`' sign. If there is no '`@`' sign, returns *string*.

`domainpart` *`string`* [Command]
: Treats *string* as an email address and returns the part following the '`@`' sign. If there is no '`@`' sign, returns empty string.

`shell` *`cmd args`* [Command]
: Runs the shell command *cmd* with the given arguments. Returns the standard output from the command. The command is invoked using `/bin/sh -c` and can contain any valid shell constructs.

The subsections below define variable names that are valid for use in each configuration statement.

3.2.3 The `include` Statement

A special statement is provided that causes inclusion of the named file. It has the following syntax:

```
include file;
```

When reading the configuration file, this statement is effectively replaced with the content of *file*. It is an error if *file* does not exist.

In site-wide configuration file, *file* can be a directory name. In this case, Mailutils will search this directory for a file with the same name as the utility being executed. If found, this file will be loaded.

It is a common to end the site-wide configuration file with an include statement, e.g.:

```
include /etc/mailutils.d;
```

This allows each particular utility to have its own configuration file. Thus, `imap4d` will read `/etc/mailutils.d/imap4d`, etc.

3.2.4 The `program` statement

Another way to configure program-specific settings is by using the `program` statement. The syntax is as follows:

```
program progname {
   ...
}
```

The `program` statement is allowed only in the site-wide configuration file. When encountered, its tag (*progname*) is compared with the name of the program being run. If two strings are the same, the statements between curly braces are stored in a temporary memory, otherwise the statement is ignored. When entire configuration file is loaded, the statements accumulated in the temporary storage are processed.

Notice the difference between this statement and a per-program configuration file loaded via an `include` statement. No matter where in the file the `program` statement is, its content will be processed after the content of the enclosing file. In the contrast, the per-program configuration file loaded via `include` is processed right where it is encountered.

3.2.5 The `logging` Statement

Syntax

```
logging {
  # Send diagnostics to syslog.
  syslog boolean;

  # Print message severity levels.
  print-severity boolean;

  # Output only messages with a severity equal to or
  # greater than this one.
  severity string;

  # Set syslog facility.
  facility name;

  # Log session ID
  session-id boolean;

  # Tag syslog messages with this string.
  tag text;
}
```

Description

The `logging` block statement configures where the diagnostic output goes and how verbose it is.

`syslog` *bool* [Configuration]
: If '`syslog`' is set to '`yes`', the diagnostics will go to syslog. Otherwise, it goes to the standard error.

The default syslog facility is determined at compile time, it can be inspected using the following command (see Section 3.20.3 [mailutils info], page 147):

```
$ mailutils info log_facility
```

`facility` *name* [Configuration]
: Use syslog facility *name*. Valid argument values are: '`user`', '`daemon`', '`auth`', '`authpriv`', '`mail`', '`cron`', '`local0`' through '`local7`' (all names case-insensitive), or a facility number.

`tag` *text* [Configuration]
: Tag syslog messages with *text*. By default, program name is used as syslog tag.

`print-severity` *bool* [Configuration]
: Print Mailutils severity name before each message.

`severity` *name* [Configuration]
: Output only messages with a severity equal to or greater than this one. Valid arguments are: '`debug`', '`info`', '`notice`', '`warning`', '`error`', '`crit`', '`alert`', '`emerg`',

`session-id` *bool* [Configuration]
: Print session ID with each diagnostic message. This is useful for programs that handle multiple user sessions simultaneously, such as `pop3d` and `imap4d`.

3.2.6 Sender Address Statements

The following two statements configure the email that `mailutils` libraries use as sender address:

`email-addr` *email* [Configuration]
: Sets the current user email address.

`email-domain` *domain* [Configuration]
: Configures the domain name that will be appended to unqualified email addresses.

In the absense of both these statements, the sender email address is constructed by concatenating the username, '`@`' sign, and the host name of the machine. If `email-domain` is present, the *domain* it defines is used instead of the hostname. Finally, if `email-addr` is present, its value is used as sender email.

3.2.7 The `debug` Statement

Syntax

```
debug {
  # Set Mailutils debugging level.
  level spec;
```

```
  # Prefix debug messages with Mailutils source locations.
  line-info bool;
}
```

Description

The '`debug`' statement controls the amount of additional debugging information output by Mailutils programs. The '`level`' statement enables additional debugging information. Its argument (*spec*) is a Mailutils debugging specification as described in Section 3.3 [debugging], page 44.

The '`line-info`' statement, when set to '`true`' causes debugging messages to be prefixed with locations in Mailutils source files where they appear. Normally, only Mailutils developers need this option.

3.2.8 The `mailbox` Statement

Syntax

```
mailbox {
  # Use specified url as a mailspool.
  mail-spool url;

  # Create mailbox url using pattern.
  mailbox-pattern pattern;

  # Default mailbox type.
  mailbox-type type;

  # Default user mail folder.
  folder dir;
}
```

Description

The `mailbox` statement configures the location, name and type of user mailboxes.

The mailbox location can be specified using `mail-spool` or `mail-pattern` statements.

`mail-spool` *path* [Configuration]

The `mail-spool` statement specifies directory that holds user mailboxes. Once this statement is given, the `libmailutils` library will assume that the mailbox of user *login* is kept in file *path/login*.

Historically, *path* can contain mailbox type prefix, e.g.: '`maildir:///var/spool/mail`', but such usage is discouraged in favor of `mailbox-pattern` statement.

`mailbox-pattern` *url* [Configuration]

The `mailbox-pattern` statement is a preferred way of configuring mailbox locations. It supersedes `mail-spool` statement.

The *url* must be a valid mailbox URL (see Chapter 2 [Mailbox], page 3), which may contain references to the '`user`' variable (see Section 3.2.2 [Variables], page 15). This variable will be expanded to the actual user name.

Optional URL parameters can be used to configure *indexed directory structure*. Such structure is a special way of storing mailboxes, which allows for faster access in case of very large number of users.

By default, all user mailboxes are stored in a single directory and are named after user login names. To find the mailbox for a given user, the system scans the directory for the corresponding file. This usually implies linear search, so the time needed to locate a mailbox is directly proportional to the ordinal number of the mailbox in the directory.

GNU Mailutils supports three types of indexed directories: '`direct`', '`reverse`', and '`hashed`'.

In direct indexed directory structure, *path* contains 26 subdirectories named with lower-case letters of Latin alphabet. The location of the user mailbox is determined using the following algorithm:

1. Take the first letter of the user name.
2. Map it to a lower-case letter using *index mapping* table. The result gives the name of a sub-directory where the mailbox is located.
3. Descend into this directory.

For example, using this algorithm, the mailbox of the user '`smith`' is stored in file *path*`/s/smith`.

If each of single-letter subdirectories contains the indexed directory structure, we have second level of indexing. In this case the file name of '`smith`''s mailbox is *path*`/s/m/smith`.

The *reverse* indexed structure uses the same principles, but the indexing letters are taken from the *end* of the user name, instead of from the beginning. For example, in the 2nd level reverse indexed structure, the '`smith`''s mailbox is located in *path*`/h/t/smith`.

Finally, the *hashed* structure consists of 256 subdirectories under *path*, named by 2-letter hex codes from '`00`' to '`FF`'. Mailboxes are stored in these subdirectories. The name of the subdirectory is computed by hashing first *level* letters of the user name. The hashing algorithm is:

1. Take next letter from the user name
2. Add its ASCII value to the hash sum.
3. Continue (1-2) until *level* letters are processed, or all letters from the file name are used, whichever occurs first.
4. Convert the computed sum modulo 256 to a hex code.

Indexed directory structures are configured using the following arguments:

type=*value*
: Specifies the type of indexing. Valid values are '`index`', for direct indexed structure, '`rev-index`' for reverse indexing, and '`hash`' for hashed structure.

param=*number*
: Specifies indexing level.

user=*string*
: Specifies indexing key. The only meaningful value, as of Mailutils version 3.19 is '`user=${user}`'.

Let's assume the traditional mail layout, in which incoming mails are stored in a UNIX mailbox named after the recipient user name and located in `/var/mail` directory. The `mailbox-pattern` for this case is:

```
mailbox-pattern "/var/mail/${user}";
```

It is entirely equivalent to specifying '`mail-spool "/var/mail"`'.

Now, if the layout is the same, but mailboxes are kept in '`maildir`' format, then the corresponding statement is:

```
mailbox-pattern "maildir:///var/mail/${user}";
```

Finally, if the mailboxes are stored in a directly-indexed directory with two levels of indexing, the URL is:

```
mailbox-pattern "maildir:///var/mail;type=index;param=2;user=${user}";
```

If neither `mailbox-pattern` nor `mail-spool` are given, the mailbox names are determined using the following algorithm:

1. If environment variable `FOLDER` is set, use its value.
2. Otherwise, if environment variable `MAIL` is set, use its value.
3. If neither of these is set, construct the mailbox name by concatenating the built-in mail spool directory name, a directory separator, and the user name.

   The built-in mail spool directory name is determined at compile time, using the '`_PATH_MAILDIR`' define from the include file `paths.h`. If this value is not defined, `/var/mail` or `/usr/spool/mail` is used.

`mailbox-type` *`type`* [Configuration]
: Specifies the type of mailboxes. By default, '`mbox`' (UNIX mailbox) is assumed. This can be changed while configuring the package by setting `MU_DEFAULT_SCHEME` configuration variable. The default value can be verified by running `mailutils info scheme`.

`folder` *`dir`* [Configuration]
: Sets user mail folder directory. Its value is used when expanding '`plus-notation`', i.e. such mailbox names as `+inbox`. The '`+`' sign is replaced by *dir*, followed by a directory separator ('`/`').

The *dir* argument can contain mailbox type prefix, e.g '`mh://Mail`'.

The default folder name is '`Mail/`'.

3.2.9 The `mime` Statement

Syntax

```
mime {
  # Define additional textual mime types.
  text-type PATTERN;
  # or
  text-type ( PATTERN-LIST );
}
```

Description

The `mime` compound statement is used by utilities that process MIME messages, in particular `mail`, `readmsg`, and `decodemail`. As of mailutils version 3.19 it contains only one statement:

`text-type` *`pattern`* [Configuration]
`text-type ( pattern-list )` [Configuration]

Defines additional patterns for recognition of textual message parts. The *pattern* is a shell globbing pattern that will be compared against the '`Content-Type`' header of a MIME message part in order to determine whether it can be treated as a text part. In second form, *pattern-list* is a comma-separated list of such patterns.

In both forms, the new patterns are appended to the built-in textual pattern list, which contains:

- text/*
- application/*shell
- application/shellscript
- */x-csrc
- */x-csource
- */x-diff
- */x-patch
- */x-perl
- */x-php
- */x-python
- */x-sh

3.2.10 The `locking` Statement

Syntax

```
locking {
  # Default locker flags.
  type default | dotlock | external | kernel | null;

  # Set the maximum number of times to retry acquiring the lock.
  retry-count number;

  # Set the delay between two successive locking attempts.
```

```
  retry-sleep arg;

  # Expire locks older than this amount of time.
  expire-timeout number;

  # Check if PID of the lock owner is active, steal the lock if it not.
  pid-check bool;

  # Use prog as external locker program.
  external-locker prog;
}
```

Description

This compound statement configures various parameters used when locking UNIX mailboxes in order to prevent simultaneous writes.

It is important to note, that locking applies only to monolithic mailboxes, i.e. mailboxes of '`mbox`' and '`dotmail`' types (see [mbox], page 3). Other mailbox types don't require locking.

`type` *`string`* [Configuration]

Set locking type. Allowed arguments are:

`default` Default locking type. As of `mailutils` version 3.19, this is equivalent to `dotlock`.

`dotlock` A '`dotlock`'-style locking. To lock a mailbox named *X* a *lock file* named *`X`*`.lock` is created. If `pid-check yes` is set, this file will contain the PID of the locking process, so that another process wishing to acquire the lock could verify if the lock is still in use.

`external` Run external program to perform locking/unlocking operations. The name of the program is given by the `external-locker` statement (see below). If it is not given, the built-in default '`dotlock`' is used.

The locker program is invoked as follows:

```
# To lock mbox:
locker -fexpire_timeout -rretry_count mbox
# To unlock it:
locker -u -fexpire_timeout -rretry_count mbox
```

Here, *expire_timeout* is the value supplied with the `expire-timeout` configuration statement, and *retry_count* is the value supplied with the `retry-count` statement (see below).

To properly interact with `mailutils`, the external locker program must use the following exit codes:

Exit code	**Meaning**
0	Success.
1	Failed due to an error.
2	Unlock requested (`-u`), but file is not locked.

3	Lock requested, but file is already locked.
4	Insufficient permissions.

See Section 3.21 [dotlock], page 167, for the description of the default external locker, shipped with mailutils.

`kernel` Use kernel locking mechanism (`fcntl`(2)).

`null` No locking at all. The statements below are silently ignored.

`retry-count` *number* [Configuration]
Number of locking attempts. The default is 10.

`retry-sleep` *seconds* [Configuration]
`retry-timeout` *seconds* [Configuration]
Time interval, in seconds, between two successive locking attempts. The default is 1 second. The `retry-timeout` statement is deprecated because of its misleading name.

`expire-timeout` *seconds* [Configuration]
Sets the expiration timeout. The existing lock file will be removed, if it was created more than this number of seconds ago. The default is 600.

`pid-check` *bool* [Configuration]
This statement can be used if locking type is set to `dotlock`. If set to `true`, it instructs the locking algorithm to check if the PID of the lock owner is still running by the time when it tries to acquire the lock. This works as follows. When the lock file is created, the PID of the creating process is written to it. If another process tries to acquire the lock and sees that the lock file already exists, it reads the PID from the file and checks if a process with that PID still exists in the process table. If it does not, the process considers the lock file to be stale, removes it and locks the mailbox.

`external-locker` *string* [Configuration]
Sets the name of the external locker program to use, instead of the default '`dotlock`'.

This statement is in effect only when used together with `type external`.

3.2.11 The `mailer` Statement

Syntax

```
mailer {
  url url;
}
```

Description

A *mailer* is a special logical entity GNU Mailutils uses for sending messages. Its internal representation is discussed in *Mailer*. The `mailer` statement configures it.

The mailer statement contains a single sub-statement:

`url` *str* [Configuration]
Set the mailer URL.

GNU Mailutils supports three types of mailer URLs, described in the table below:

smtp://[*user*[:*pass*][;auth=*mech*,...]@]*host*[:*port*][;*params*]
smtps://[*user*[:*pass*][;auth=*mech*,...]@]*host*[:*port*][;*params*]
: Send messages using SMTP protocol. See Section 2.3 [SMTP Mailboxes], page 5, for a detailed description of the URL and its parts.

sendmail[://*progname*]
: Use sendmail-compatible program *progname*. *Sendmail-compatible* means that the program must support following command line options:

  `-oi` Do not treat '.' as message terminator.

  `-f addr` Use *addr* as the sender address.

  `-t` Get recipient addresses from the message.

  See Section 2.4 [sendmail], page 6, for details.

prog://*progname*?*query*
: A *prog* mailer. This is a generalization of '`sendmail`' mailer that allows to use arbitrary external programs as mailers.

  It is described in detain in Section 2.4 [prog], page 6.

3.2.12 The `acl` Statement

Syntax

```
acl {
  # Allow connections from this IP address.
  allow [from] ip;

  # Deny connections from this IP address.
  deny [from] ip;

  # Log connections from this IP address.
  log [from] ip [string];

  /* Execute supplied program if a connection from this
   IP address is requested. */
  exec [from] ip program;

  /* Use program to decide whether to allow connection
     from ip. */
  ifexec [from] ip program;
}
```

Description

The ACL statement defines an *Access Control List*, a special structure that controls who can access the given Mailutils resource.

The `acl` block contains a list of access controls. Each control can be regarded as a function that returns a tree-state value: '`True`', '`False`' and '`Don't know`'. When a remote

party connects to the server, each of controls is tried in turn. If a control returns '`False`', access is denied. If it returns '`True`', access is allowed. If it returns '`Don't know`', then the next control is tried. It is unclear whether to allow access if the last control in list returned '`Don't know`'. GNU Mailutils 3.19 issues a warning message and allows access. This default may change in future versions. Users are advised to write their ACLs so that the last control returns a definite answer (either `True` or `False`).

In the discussion below, wherever *cidr* appears as an argument, it can be replaced by any of:

- An IPv4 address in dotted-quad notation.
- An IPv6 address in numeric notation
- A CIDR in the form '*ip/mask*', where *ip* is an IP address (either IPv4 or IPv6), and *mask* is the network mask.
- A symbolic host name.
- A word '`any`', which matches any IP address.

The following controls are understood:

`allow` [*from*] *cidr* [Configuration]
: Allow connections from IP addresses matching this *cidr* block.

`deny` [*from*] *cidr* [Configuration]
: Deny connections from IP addresses matching this *cidr* block.

`ifexec` [*from*] *cidr program* [Configuration]
: When a connection from the *cidr* block is requested, execute the program *program*. If its exit code is '`0`', then allow connection. Otherwise, deny it.

  The *program* argument undergoes variable expansion and word splitting. The following variables are defined:

  `aclno` Ordinal number of the control in the ACL. Numbers begin from '`1`'.

  `family` Connection family. Mailutils version 3.19 supports the following families: '`AF_INET`', '`AF_INET6`' and '`AF_UNIX`'.

  `address` Remote IP address (for '`AF_INET`' and '`AF_INET6`') or socket name (for '`AF_UNIX`'). Notice that most Unixes return empty string instead of the '`AF_UNIX`' socket name, so do not rely on it.

  `port` Remote port number (for '`AF_INET`' and '`AF_INET6`').

`exec` [*from*] *cidr program* [Configuration]
: If a connection from the *cidr* block is requested, execute the given *program*. Do not wait for it to terminate, and ignore its exit code. The *program* is subject for variable expansion as in '`ifexec`'.

The following two controls are provided for logging purposes and as a means of extensions. They always return a '`Don't know`' answer, and therefore should not be used at the end of an ACL:

`log` [*from*] `cidr` [`string`] [Configuration]

Log connections from addresses in this *cidr*. The `MU_DIAG_INFO` channel is used. If the logging goes to syslog, it is translated to the `LOG_INFO` priority.

If *string* is not given, the format of the log entry depends on the connection family, as described in the table below:

{AF_INET *ip*:*port*}
: For inet IPv4 connections. The variables *ip* and *port* are replaced by the remote IP address and port number, correspondingly.

{AF_UNIX}
: For connections over UNIX sockets. The socket name, if available, may be printed before the closing curly brace.

If *string* is supplied, it undergoes variable expansions as described for the '`ifexec`'.

For example, the following ACL makes a Mailutils server log every incoming connection:

```
acl {
   log from any "Connect from ${address}";
   ...
}
```

This was the default behavior for the versions of Mailutils up to '`1.2`', so if you got used to its logs you might wish to add the above in your configuration files.

`exec` [*from*] `cidr` `program` [Configuration]

If a connection from the *cidr* block is requested, execute the given *program*. Do not wait for it to terminate, and ignore its exit code.

3.2.13 The `tcp-wrappers` Statement

Syntax

```
tcp-wrappers {
  # Enable TCP wrapper access control.
  enable bool;

  # Set daemon name for TCP wrapper lookups.
  daemon name;

  # Use file for positive client address access control.
  allow-table file;

  # Use file for negative client address access control.
  deny-table file;
}
```

Description

The `tcp-wrappers` statements provides an alternative way to control accesses to the resources served by GNU Mailutils. This statement is enabled if Mailutils is compiled with TCP wrappers library `libwrap`.

Access control using TCP wrappers is based on two files, called *tables*, containing access rules. There are two tables: the *allow table*, usually stored in file `/etc/hosts.allow`, and the *deny table*, kept in file `/etc/hosts.deny`. The rules in each table begin with an identifier called *daemon name*. A utility that wishes to verify a connection, selects the entries having its daemon name from the allow table. A connection is allowed if it matches any of these entries. Otherwise, the utility retrieves all entries with its daemon name from the deny table. If any of these matches the connection, then it is refused. Otherwise, if neither table contains matching entries, the connection is allowed.

The description of a TCP wrapper table format lies outside the scope of this document. Please, see Section "ACCESS CONTROL FILES" in *hosts_access(5) man page*, for details.

`enable` *bool* [Configuration]
Enable access control using TCP wrappers. It is on by default.

`daemon` *name* [Configuration]
Set daemon name for TCP wrapper lookups. By default, the name of the utility is used. E.g. `imap4d` uses '`imap4d`' as the daemon name.

`allow-table` *file* [Configuration]
Use *file* as allow table. By default, `/etc/hosts.allow` is used.

`deny-table` *file* [Configuration]
Use *file* as negative table. By default, `/etc/hosts.deny` is used.

3.2.14 Server Settings

GNU Mailutils offers several server applications: `pop3d`, `imap4d`, `comsatd`, to name a few. Being quite different in their purpose, they are very similar in some aspects of their architecture. First of all, they all support two operating modes: *daemon*, where a program disconnects from the controlling terminal and works in background, and *inetd*, where it remains in foreground and communicates with the remote party via standard input and output streams. Secondly, when operating as daemons, they listen to a preconfigured set of IP addresses and ports, reacting to requests that arrive.

To configure these aspects of functionality, GNU Mailutils provides *Server Configuration Settings*, which is describes in this subsection.

3.2.14.1 General Server Configuration

Syntax:

```
# Set daemon mode.
mode 'inetd|daemon';

# Run in foreground.
foreground bool;

# Maximum number of children processes to run simultaneously.
max-children number;
```

```
# Store PID of the master process in file.
pidfile file;

# Default port number.
port portspec;

# Set idle timeout.
timeout time;
```

Description: These statements configure general server-related issues.

`mode string;` [Configuration]

Set operation mode of the server. Two operation modes are supported:

daemon
: Run as a standalone daemon, disconnecting from the controlling terminal and continuing to run in the background. In this case, it is the server that controls what IP addresses and ports to listen on, who is allowed to connect and from where, how many clients are allowed to connect simultaneously, etc. Most remaining configuration statements are valid only in the daemon mode.

  This is the preferred mode of operation for GNU Mailutils servers.

inetd
: Operate as a subprocess of UNIX internet super-server program, `inetd`. See Section "Internet super-server" in *inetd(8) man page*, for a detailed description of the operation of `inetd` and its configuration. In this case it is `inetd` that controls all major connectivity aspects. The Mailutils server program communicates with it via standard input and output streams.

  For historical reasons, this mode is the default, if no `mode` statement is specified. This will change in the future.

`foreground bool;` [Configuration]

[daemon mode only]
Do not disconnect from the controlling terminal and remain in the foreground.

`max-children number;` [Configuration]

[daemon mode only]
Set maximum number of child processes allowed to run simultaneously. This equals the number of clients that can use the server simultaneously.

The default is 20 clients.

`pidfile file;` [Configuration]

After startup, store the PID of the main server process in *file*. When the process terminates, the file is removed. As of version 3.19, GNU Mailutils servers make no further use of this file. It is intended for use by automated startup scripts and controlling programs (e.g. see *GNU Pies Manual*).

`port` *portspec;* [Configuration]

[daemon mode only]
Set default port to listen to. The *portspec* argument is either a port number in decimal, or a symbolic service name, as listed in `/etc/services` (see Section "Internet network services list" in *services(5) man page*).

`timeout` *time;* [Configuration]
Sets maximum idle time out in seconds. If a client does not send any requests during *time* seconds, the child process terminates.

3.2.14.2 The `server` Statement

Syntax:

```
server ipaddr[:port] {
  # Run this server as a single process.
  single-process bool;

  # Log the session transcript.
  transcript bool;

  # Set idle timeout.
  timeout time;

  # Size of the queue of pending connections
  backlog <number: callback>;

  # Kind of TLS encryption to use for this server.
  tls-mode 'no'|'ondemand'|'required'|'connection';

  tls {
    # Specify SSL certificate file.
    ssl-certificate-file string;
    # Specify SSL certificate key file.
    ssl-key-file file;
    # Specify trusted CAs file.
    ssl-ca-file file;
    # Set the priorities to use on the ciphers, methods, etc.
    ssl-priorities string;
    # Set timeout for I/O operations during TLS handshake (seconds).
    handshake-timeout n;
  }

  # Set server specific ACLs.
  acl { /* See [ACL Statement], page 25. */ };
}
```

Description:

The `server` block statement configures a single TCP or UDP server. It takes effect only in daemon mode (see [server mode], page 29). The argument to this statement specifies the IP address, and, optionally, the port, to listen on for requests. The *ipaddr* part is either an IPv4 address in dotted-quad form, or a IPv6 address enclosed in square brackets, or a symbolic host name which can be resolved to such an address. Specifying '`0.0.0.0`' as the *ipaddr* means listen on all available network interfaces. The *port* argument is either a port number in decimal, or a symbolic service name, as listed in `/etc/services` (see Section "Internet network services list" in *services(5) man page*). If *port* is omitted, Mailutils uses the port set by `port` statement (see Section 3.2.14.1 [General Server Configuration], page 28), or, in its absence, the default port number, which depends on a server being used (e.g. 110, for `pop3d`, 143, for `imap4d`, etc.).

Any number of `server` statements may be specified in a single configuration file, allowing to set up the same service on several IP addresses and/or port numbers, and with different configurations.

Statements within the `server` block statement configure this particular server.

`single-process` *`bool;`* [Configuration]
: If set to true, this server will operate in single-process mode. This mode is intended for debugging only, do not use it on production servers.

`transcript` *`bool;`* [Configuration]
: Enable transcript of the client-server interaction. This may generate excessive amounts of logging, which in turn may slow down the operation considerably.

  Session transcripts are useful in fine-tuning your configurations and in debugging. They should be turned off on most production servers.

`timeout` *`time;`* [Configuration]
: Set idle timeout for this server. This overrides the global timeout settings (see Section 3.2.14.1 [General Server Configuration], page 28).

`backlog` *`number;`* [Configuration]
: Configures the size of the queue of pending connections

`tls-mode` *`mode;`* [Configuration]
: Configure the use of TLS encryption. The *mode* argument is one of the following:

  no
  : TLS is not used. The corresponding command (`STLS`, for POP3, `STARTTLS`, for `IMAP4`) won't be available even if the TLS configuration is otherwise complete.

  ondemand
  : TLS is initiated when the user issues the appropriate command. This is the default when TLS is configured.

  required
  : Same as above, but the use of TLS is mandatory. The authentication state is entered only after TLS negotiation has succeeded.

  connection
  : TLS is always forced when the connection is established. For `pop3d` this means using POP3S protocol (or IMAP4S, for `imap4d`).

`tls { ... }` [Configuration]

The `tls` statement configures SSL certificate and key files, as well as other SSL settings for use in this server. It is used when `tls-mode` is set to any of the following values: `ondemand`, `required`, `connection`.

If `tls-mode` is set to any of the values above and `tls` section is absent, settings from the global `tls` section will be used. In this case, it is an error if the global `tls` section is not defined.

See Section 3.2.21 [tls statement], page 42, for a discussion of its syntax.

`acl` [Configuration]

This statement defines a per-server Access Control List. Its syntax is as described in [ACL Statement], page 25. Per-server ACLs complement, but not override, global ACLs, i.e. if both global ACL and per-server ACL are used, the connection is allowed only if both of them allow it, and is denied if any one of them denies it.

3.2.15 The `auth` Statement

Syntax

```
auth {
  # Set a list of modules for authentication.
  authentication module-list;

  # Set a list of modules for authorization.
  authorization module-list;
}
```

Description

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 one of them succeeds or the list is exhausted. In the 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.

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 `auth` statement configures authentication and authorization.

`authorization` *`module-list`* [Configuration]

Define a sequence of modules to use for authorization. Modules will be tried in the same order as listed in *module-list*.

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 a SQL database. A separate configuration statement, `sql`, is used to configure it (see Section 3.2.19 [sql statement], page 37).

virtdomain
User credentials are retrieved from a "virtual domain" user database. Virtual domains are configured using `virtdomain` statement (see Section 3.2.17 [virtdomain statement], page 34).

radius User credentials are retrieved using RADIUS. See Section 3.2.18 [radius statement], page 35, for a detailed description on how to configure it.

ldap User credentials are retrieved from an LDAP database. See Section 3.2.20 [ldap statement], page 40, for an information on how to configure it.

Unless overridden by `authorization` statement, the default list of authorization modules is:

1. generic
2. system
3. pam
4. sql
5. virtual
6. radius
7. ldap

`authentication` *`module-list`* [Configuration]

Define a sequence of modules to use for authentication. Modules will be tried in the same order as listed in *module-list*.

The following table lists modules available for use in *module-list*:

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 a SQL database using query supplied by `getpass` statement (see Section 3.2.19 [sql statement], page 37).

pam The user is authenticated via pluggable authentication module (PAM). The PAM service name to be used is configured in `pam` statement (see Section 3.2.16 [pam statement], page 34).

radius The user is authenticated on a remote RADIUS server. See Section 3.2.18 [radius statement], page 35.

ldap The user is authenticated using LDAP. See Section 3.2.20 [ldap statement], page 40.

Unless overridden by `authentication` statement, the list of authentication modules is the same as for `authorization`, i.e.:

1. generic
2. system
3. pam
4. sql
5. virtual
6. radius
7. ldap

3.2.16 PAM Statement

Syntax

```
pam {
  # Set PAM service name.
  service text;
}
```

Description

The `pam` statement configures PAM authentication. It contains a single sub-statement:

`service` *`text`* [Configuration]
Define service name to look for in PAM configuration. By default, the base name of the Mailutils binary is used.

This statement takes effect only if '`pam`' is listed in `authentication` statement (see Section 3.2.15 [auth statement], page 32).

3.2.17 The `virtdomain` Statement

Syntax

```
virtdomain {
  # Name of the virtdomain password directory.
  passwd-dir dir;
}
```

Description

Virtual mail domains make it possible to handle several mail domains each having a separate set of users, on a single server. The domains are completely independent of each other, i.e. the same user name can be present in several domains and represent different users.

When authenticating to a server with virtual domain support enabled, users must supply their user names with domain parts. The server strips off the domain part and uses it as a name of UNIX-format password database file, located in the *domain password directory*. The latter is set using `passwd-dir` statement.

`passwd-dir` *dir* [Configuration]

Set virtual domain password directory.

For example, when authenticating user '`smith@example.com`', the server will use password file named *dir*`/example.com`. This file must be in UNIX passwd format (see Section "password file" in *passwd(5) man page*), with encrypted passwords stored in it (as of GNU Mailutils version 3.19, there is no support for shadow files in virtual password directories, although this is planned for future versions). Here is an example record from this file:

```
smith:Wbld/G2Q2Le2w:1000:1000:Email Account:/var/mail/domain/smith:/dev/null
```

Notice, that it must contain user names without domain parts.

The `pw_dir` field (the 6th field) is used to determine the location of the maildrop for this user. It is defined as *pw_dir*`/INBOX`. In our example, the maildrop for user '`smith`' will be located in file `/var/mail/domain/smith`.

If user did not supply his domain name, or if no matching record was found in the password file, or if the file matching the domain name does not exist, then GNU Mailutils falls back to alternative method. First, it tries to determine the IP address of the remote party. Then the domain name corresponding to that address is looked up in the DNS system. Finally, this domain name is used as a name of the password file.

3.2.18 The `radius` Statement

Syntax

```
radius {
  # Set radius configuration directory.
  directory dir;
  # Radius request for authorization.
  auth request;
  # Radius request for getpwnam.
  getpwnam request;
  # Radius request for getpwuid.
  getpwuid request;
}
```

Description

The `radius` block statement configures RADIUS authentication and authorization.

Mailutils uses GNU Radius library, which is configured via `raddb/client.conf` file (see Section "Client Configuration" in *GNU Radius Reference Manual*). Its exact location depends on configuration settings that were used while compiling GNU Radius. Usually it is `/usr/local/etc`, or `/etc`. This default can also be changed at run time using `directory` statement:

`directory` *dir* [Configuration]
: Set full path name to the GNU Radius configuration directory.

It authorization is used, the Radius dictionary file must declare the the following attributes:

Attribute	**Type**	**Description**
GNU-MU-User-Name	string	User login name
GNU-MU-UID	integer	UID
GNU-MU-GID	integer	GID
GNU-MU-GECOS	string	GECOS
GNU-MU-Dir	string	Home directory
GNU-MU-Shell	string	User shell
GNU-MU-Mailbox	string	User mailbox
GNU-MU-Quota	integer	Mail quota (in bytes)

A dictionary file with appropriate definitions is included in the Mailutils distribution: `examples/config/mailutils.dict`. This file is not installed by default, you will have to manually copy it to the GNU Radius `raddb/dict` directory and include it in the main dictionary file `raddb/dictionary` by adding the following statement:

```
$INCLUDE dict/mailutils.dict
```

Requests to use for authentication and authorization are configured using three statements: `auth`, `getpwnam` and `getpwuid`. Each statement takes a single argument: a string, containing a comma-separated list of assignments. An assignment specifies a particular *attribute-value pair* (see Section "Overview" in *GNU Radius Reference Manual*) to send to the server. The left-hand side of the assignment is a symbolic attribute name, as defined in one of Radius dictionaries (see Section "dictionary file" in *GNU Radius Reference Manual*). The value is specified by the right-hand side of assignment. For example:

```
"Service-Type = Authenticate-Only, NAS-Identifier = \"mail\""
```

The assignment may contain references to the following variables (see Section 3.2.2 [Variables], page 15):

user
: The actual user name (for `auth` and `getpwnam`), or user ID (for `getpwuid`). For example:

```
User-Name = ${user}
```

passwd
: User password. For examples:

```
User-Password = ${passwd}
```

`auth` *pairlist* [Configuration]

Specifies the request to be sent to authenticate the user. For example:

```
auth "User-Name = ${user}, User-Password = ${passwd}";
```

The user is authenticated only if this request returns `Access-Accept` (see Section "Authentication Requests" in *GNU Radius Reference Manual*). Any returned attribute-value pairs are ignored.

`getpwnam` *pairlist* [Configuration]

Specifies the request that returns user information for the given user name. For example:

```
getpwnam "User-Name = ${user}, State = getpwnam, "
         "Service-Type = Authenticate-Only";
```

If the requested user account exists, the Radius server must return `Access-Accept` packet with the following attributes: `GNU-MU-User-Name`, `GNU-MU-UID`, `GNU-MU-GID`, `GNU-MU-GECOS`, `GNU-MU-Dir`, `GNU-MU-Shell`.

The attributes `GNU-MU-Mailbox` and `GNU-MU-Quota` are optional.

If `GNU-MU-Mailbox` is present, it must contain a valid mailbox URL (see Chapter 2 [Mailbox], page 3). If `GNU-MU-Mailbox` is not present, Mailutils constructs the mailbox name using the settings from the `mailbox` configuration statement (see [Mailbox Statement], page 19), or built-in defaults, if it is not present.

If `GNU-MU-Quota` is present, it specifies the maximum mailbox size for this user, in bytes. In the absence of this attribute, mailbox size is unlimited.

`getpwuid` *pairlist* [Configuration]

Specifies the request that returns user information for the given user ID. In *pairlist*, the '`user`' macro-variable is expanded to the numeric value of ID. For example:

```
getpwuid "User-Name = ${user}, State = getpwuid, "
         "Service-Type = Authenticate-Only";
```

The reply to `getpwuid` request is the same as to `getpwnam` request (see above).

3.2.19 The `sql` Statement

Syntax

```
sql {
  # Set SQL interface to use.
  interface 'mysql|odbc|postgres';
  # SQL server host name.
  host arg;
  # SQL user name.
  user arg;
  # Password for the SQL user.
  passwd arg;
  # SQL server port.
  port arg;
  # Database name.
```

```
    db arg;
    # Type of password returned by getpass query.
    password-type 'plain | hash | scrambled';
    # Set a field-map for parsing SQL replies.
    field-map list;
    # SQL query returning the user's password.
    getpass query;
    # SQL query to use for getpwnam requests.
    getpwnam query;
    # SQL query to use for getpwuid requests.
    getpwuid query;
  }
```

Description

The `sql` statement configures access credentials to SQL database and the queries for authentication and authorization.

GNU Mailutils supports three types of SQL interfaces: MySQL, PostgreSQL and ODBC. The latter is a standard API for using database management systems, which can be used to communicate with a wide variety of DBMS.

`interface` *type* [Configuration]

Configures type of DBMS interface. Allowed values for *type* are:

mysql	Interface with a MySQL server (`http://www.mysql.org`).
odbc	Use ODBC interface. See `http://www.unixodbc.org`, for a detailed description of ODBC configuration.
postgres	Interface with a PostgreSQL server (`http://www.postgres.org`).

The database and database access credentials are configured using the following statements:

`host` *arg* [Configuration]

The host running the SQL server. The value can be either a host name or an IP address in dotted-quad notation, in which case an INET connection is used, or a full pathname to a file, in which case a connection to UNIX socket is used.

`port` *arg* [Configuration]

TCP port the server is listening on (for INET connections). This parameter is optional. Its default value depends on the type of database being used.

`db` *arg;* [Configuration]

Name of the database.

`user` *arg* [Configuration]

SQL user name.

`passwd` *arg;* [Configuration]

Password to access the database.

`password-encryption` *arg*; [Configuration]

Defines type of encryption used by the password returned by `getpass` query (see below). Possible arguments are:

plain
: Password is in plain text.

crypt
hash
: Password is encrypted by system `crypt` function (see Section "crypt" in *crypt(3) man page*).

scrambled
: Password is encrypted by MySQL `password` function.

`getpwnam` *query* [Configuration]

Defines SQL query that returns information about the given user. The *query* is subject to variable expansion (see Section 3.2.2 [Variables], page 15). The only variable defined is '`$user`', which expands to the user name.

The query should return a single row with the following columns:

name
: User name.

passwd
: User password.

uid
: UID of the user.

gid
: GID of the primary group.

gecos
: Textual description of the user.

dir
: User's home directory

shell
: User's shell program.

The following columns are optional:

mailbox
: Full pathname of the user's mailbox. If not returned or NULL, the mailbox is determined using the default algorithm (see Chapter 2 [Mailbox], page 3).

quota
: Upper limit on the size of the mailbox. The value is either an integer number optionally followed by one of the usual size suffixes: '`K`', '`M`', '`G`', or '`T`' (case-insensitive).

`getpwuid` *query* [Configuration]

Defines SQL query that returns information about the given UID. The *query* is subject to variable expansion (see Section 3.2.2 [Variables], page 15). The only variable defined is '`$user`', which expands to the UID.

The query should return a single row, as described for `getpwnam`.

`getpass` *query* [Configuration]

Defines SQL query that returns the password of the given user. The *query* is subject to variable expansion (see Section 3.2.2 [Variables], page 15). The only variable defined is '`$user`', which expands to the user name.

The query should return a row with a single column, which gives the password. The password can be encrypted as specified by the `password-encryption` statement.

`field-map` *list* [Configuration]

Defines a translation map for column names. The *list* is a list of mappings. Each mapping is a string '*name=column*', where *name* is one of the names described in [getpw column names], page 39, and *column* is the name of the column in the returned row that should be used instead. The effect of this statement is similar to that of SQL `AS` keyword. E.g. the statement

```
field-map (uid=user_id);
```

has the same effect as using '`SELECT user_id AS uid`' in the SQL statement.

3.2.20 The `ldap` Statement

Syntax

```
ldap {
  # Enable LDAP lookups.
  enable bool;
  # Set URL of the LDAP server.
  url url;
  # Base DN for LDAP lookups.
  base string;
  # DN for accessing LDAP database.
  binddn string;
  # Password for use with binddn.
  passwd string;
  # Use TLS encryption.
  tls bool;
  # Set LDAP debugging level.
  debug number;
  # Set a field-map for parsing LDAP replies.
  field-map list;
  # LDAP filter to use for getpwnam requests.
  getpwnam string;
  # LDAP filter to use for getpwuid requests.
  getpwuid filter;
}
```

Description

The `ldap` statement configures the use of LDAP for authentication.

`enable` *bool* [Configuration]

Enables LDAP lookups. If absent, '`enable On`' is assumed.

`url` *url* [Configuration]

Sets the URL of the LDAP server.

`base` *string* [Configuration]

Defines base DN for LDAP lookups.

`binddn` *string* [Configuration]
Defines the DN for accessing LDAP database.

`passwd` *string* [Configuration]
Password for use when binding to the database.

`tls` *bool* [Configuration]
Enable the use of TLS when connecting to the server.

`debug` *number* [Configuration]
Set LDAP debug level. Please refer to the OpenLDAP documentation, for allowed *number* values and their meaning.

`field-map` *map* [Configuration]
Defines a map for parsing LDAP replies. The *map* is a list of mappings[1]. Each mapping is '*field*`=`*attr*', where *attr* is the name of the LDAP attribute and *field* is a field name that declares what information that attribute carries. Available values for *field* are:

name	User name.
passwd	User password.
uid	UID of the user.
gid	GID of the primary group.
gecos	Textual description of the user.
dir	User's home directory
shell	User's shell program.

The default mapping is

```
("name=uid",
 "passwd=userPassword",
 "uid=uidNumber",
 "gid=gidNumber",
 "gecos=gecos",
 "dir=homeDirectory",
 "shell=loginShell")
```

`getpwnam` *string* [Configuration]
Defines the LDAP filter to use for '`getpwnam`' requests. The default is:

```
(&(objectClass=posixAccount) (uid=$user))
```

`getpwuid` *string* [Configuration]
Defines the LDAP filter to use for '`getpwuid`' requests. The default filter is:

```
(&(objectClass=posixAccount) (uidNumber=$user))
```

[1] For backward compatibility, *map* can be a string containing colon-delimited list of mappings. Such usage is, however, deprecated.

3.2.21 The `tls` Statement

Syntax

```
tls {
  # Specify SSL certificate file.
  ssl-certificate-file string;
  # Specify SSL certificate key file.
  ssl-key-file file;
  # Specify trusted CAs file.
  ssl-ca-file file;
  # Set the priorities to use on the ciphers, methods, etc.
  ssl-priorities string;
  # Set timeout for I/O operations during TLS handshake (seconds).
  handshake-timeout n;
}
```

Description

The '`tls`' statement configures TLS parameters to be used by servers. It can appear both in the global scope and in server scope. Global tls settings are applied for servers that are declared as supporting TLS encryption, but lack the '`tls`' substatement.

`ssl-certificate-file` *string* [Configuration]
: Specify SSL certificate file.

`ssl-key-file` *file* [Configuration]
: Specify SSL certificate key file.

`ssl-ca-file` *file* [Configuration]
: Specify the trusted certificate authorities file.

`ssl-priorities` *string* [Configuration]
: Set the priorities to use on the ciphers, key exchange methods, MACs and compression methods.

`handshake-timeout` *n* [Configuration]
: Set the timeout (in seconds) for I/O operations during TLS handshake. Default value is 10 seconds.

3.2.22 The `tls-file-checks` Statement

Syntax

```
tls-file-checks {
  # Configure safety checks for SSL key file.
  key-file list;
  # Configure safety checks for SSL certificate.
  cert-file list;
  # Configure safety checks for SSL CA file.
  ca-file list;
}
```

Description

This section configures security checks applied to the particular SSL configuration files in order to decide whether it is safe to use them.

`key-file` *list* [Configuration]

Configure safety checks for SSL key file. Elements of the *list* are names of individual checks, optionally prefixed with '`+`' to enable or '`-`' to disable the corresponding check. Valid check names are:

none	Disable all checks.
all	Enable all checks.
gwrfil	Forbid group writable files.
awrfil	Forbid world writable files.
grdfil	Forbid group readable files.
ardfil	Forbid world writable files.
linkwrdir	Forbid symbolic links in group or world writable directories.
gwrdir	Forbid files in group writable directories.
awrdir	Forbid files in world writable directories,

`cert-file` *list* [Configuration]

Configure safety checks for SSL certificate. See `key-file` for a description of *list*.

`ca-file` *list* [Configuration]

Configure safety checks for SSL CA file. See `key-file` for a description of *list*.

3.2.23 The `gsasl` Statement

==========

Editor's note:

This node is to be written.

==========

Syntax

```
gsasl {
  # Name of GSASL password file.
  cram-passwd file;
  # SASL service name.
  service string;
  # SASL realm name.
  realm string;
  # SASL host name.
  hostname string;
  # Anonymous user name.
  anonymous-user string;
}
```

3.3 Debugging

`Mailutils` debugging output is controlled by a set of levels, each of which can be set independently of others. Each debug level consists of a *category name*, which identifies the part of `Mailutils` for which additional debugging is desired, and a level number, which tells `Mailutils` how verbose should its output be.

Valid debug levels are:

`error` Displays error conditions which are normally not reported, but passed to the caller layers for handling.

`trace0` through `trace9`
Ten levels of verbosity, '`trace0`' producing less output, '`trace9`' producing the maximum amount of output.

`prot` Displays network protocol interaction, where applicable.

Implementation and applicability of each level differs between various categories. The full list of categories is available in file `libmailutils/diag/debcat` in the Mailutils source tree. Most useful categories and levels implemented for them are discussed later in this article.

3.3.1 Level Syntax

Debug levels can be set either from the command line, by using the `--debug-level` command line option, or from the configuration file, using the '`.debug.level`' statement. In both cases, the level is specified as a list of individual levels, delimited with semicolons. Each individual level can be specified as:

!*category* Disables all levels for the specified *category*.

category Enables all levels for the specified *category*.

category.level
For the given *category*, enables all levels from '`error`' to *level*, inclusive.

category.=level
Enables only the given *level* for this *category*.

category.!level
Disables all levels from '`error`' to *level*, inclusive, for this *category*.

category.!=level
Disables only the given *level* in this *category*.

category.levelA-levelB
Enables all levels in the range from *levelA* to *levelB*, inclusive.

category.!levelA-levelB
Disables all levels in the range from *levelA* to *levelB*, inclusive.

Additionally, a comma-separated list of level specifications is allowed after the dot. For example, the following specification:

```
acl.prot,!=trace9,!trace2
```

enables in category '`acl`' all levels, except '`trace9`', '`trace0`', '`trace1`', and '`trace2`'.

3.3.2 BNF

The following specification in Backus-Naur form describes formally the Mailutils debug level:

```
<debug-spec> ::= <level-spec> | <debug-level-list>
<debug-level-list> ::= <debug-level> |
                       <debug-level-list> ";" <debug-level>
<debug-level> ::= <category> | "!" <category> |
                  <category> "." <level-list>
<level-list> ::= <level-spec> | <level-list> "," <level-spec>
<level-spec> ::=  <level> | <negate-level>
<negate-level> ::= "!" <level>
<level> ::= <level-number> | "=" <level-number> |
            <level-number> "-" <level-number>
<level-number> ::= "error" | "trace0" | "trace1" | "trace2" | "trace3" |
                   "trace4" | "trace5" | "trace6" | "trace7" |
                   "trace8" | "trace9" | "prot"
```

3.3.3 Debugging Categories

`all` [Category]

This is not a category as such, but rather a convenient shortcut for setting all categories. For example:

`all` Sets all levels in all debugging categories.

`all.trace6`
Sets levels up to and including `trace6` in all debugging categories.

`all.!=prot;!mailer;auth.prot`
`all.!=prot;!mailer;auth`
Sets all levels except `prot` in all categories, excepting `mailer` and `auth`. The `mailer` category is disabled. All levels are set in `auth`.

`acl` [Category]

This category enables debugging of Access Control Lists. Available levels are:

`error` As usual, displays errors, not directly reported otherwise.

`trace0` Basic tracing of ACL processing.

`trace9` Traces the process of matching the ACL conditions.

`config` [Category]

This category affects configuration parser and/or lexical analyzer. The following levels are supported:

`trace0` Minimal information about configuration statements.

`trace2` Trace lexical structure of the configuration files.

`trace7` Trace execution of the configuration parser.

Due to its specific nature, this category cannot be enabled from the configuration file. A special hook is provided to facilitate debugging the configuration parser, namely, a pragmatic comment in form:

```
#debug=debug-level-list
```

causes *debug-level-list* to be parsed as described above. Thus, to force debugging of the configuration parser, one would add the following line at the very beginning of the configuration file:

```
#debug=config.trace7
```

`mailbox` [Category]

Operations over mailboxes. This module supports the following levels: '`error`', '`trace0`', '`trace1`', and '`prot`'. The latter is used by remote mailbox support libraries.

`auth` [Category]

Enables debugging information about authentication and authorization. This category supports the following levels: '`error`', '`trace0`', '`trace1`', and '`trace2`'.

In level '`trace0`', user data are reported along with the *data source* they were obtained from. The output may look like this:

```
pop3d: source=system, name=gray, passwd=x, uid=120, gid=100,
gecos=Sergey Poznyakoff, dir=/home/gray, shell=/bin/bash,
mailbox=/var/mail/gray, quota=0, change_uid=1
```

In the '`trace1`' level, additional flow traces are displayed.

In the level '`trace2`', a detailed flow trace is displayed, which looks like the following:

```
pop3d: Trying generic...
pop3d: generic yields 38=Function not implemented
pop3d: Trying system...
pop3d: system yields 0=Success
pop3d: Trying generic...
pop3d: generic yields 4135=Authentication failed
pop3d: Trying system...
pop3d: system yields 0=Success
```

`mailer` [Category]

Debugs mailer operations. The following levels are supported:

`error` Displays mild error conditions.

`trace0` Traces mailer operations in general: displays what part of the message is being sent, etc.

`trace6` When used together with '`prot`', displays security-sensitive information (such as passwords, user keys, etc). in plaintext. By default, such information is replaced with asterisks to reduce the possibility of security compromise.

`trace7` When used together with '`prot`', displays the *payload* information as it is being sent. The *payload* is the actual message contents, i.e. the part of

SMTP transaction that goes after the '`DATA`' command and which ends with a terminating dot line. Setting this level can generate huge amounts of information.

`prot` For SMTP mailer: outputs transcripts of SMTP sessions.

Note: Unless in a very secure environment, it is advised to avoid using level settings such as mailer.prot or mailer (without explicit level part), because the resulting output tends to be extremely copious and reveals sender private and security-sensitive data. If you wish to trace SMTP session flow, use '`mailer.=prot`' or at least '`mailer.prot,!trace6`'.

`server` [Category]
This category provides debugging information for Mailutils IP server objects. It supports the '`error`' and '`trace0`' levels.

`folder` [Category]
This category controls debugging information shown for operations related to Mailutils folders.

`remote` [Category]
The remote category is used by `imap4d` and `pop3d` servers to request showing additional information in the session transcripts. This category takes effect only when the `transcript` configuration variable is set. Valid levels are:

`trace6` Show security-sensitive information (user passwords, etc.)

`trace7` Show payload information

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

==========

Editor's note:

The information in this node may be obsolete or otherwise inaccurate. This message will disappear, once this node revised.

==========

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

The behavior of both programs is affected by the following configuration file statements:

Statement	Reference
debug	See Section 3.2.7 [debug statement], page 18.
tls	See Section 3.2.21 [tls statement], page 42.
mailbox	See Section 3.2.8 [mailbox statement], page 19.
locking	See Section 3.2.10 [locking statement], page 22.

frm

The `frm` utility outputs a header information of the selected messages in a mailbox. By default, `frm` reads 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 following command line options alter the behavior of the program:

`-d`
`--debug` Enable debugging output.

`-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 name of the user 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.

The option `-f` (`--file`) instructs `from` 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 output.

`-s` *string*
`--sender=`*string*
Prints only mail with '`From:`' header containing the supplied string.

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

3.5 `mail` — Send and Receive Mail

==========

Editor's note:

The information in this node may be obsolete or otherwise inaccurate. This message will disappear, once this node revised.

==========

`Mail` is an enhanced version of POSIX `mailx` program. The program operates in two modes: *read* and *send*.

`Mail` enters *send* mode when at least one email address was specified in its 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 3.5.4 [Composing Mail], page 67, 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 the user to read and manipulate the contents of the user system mailbox. Use the `--file` (`-f`) option to specify another mailbox name. For more detail, see Section 3.5.2 [Reading Mail], page 52.

In addition to the Mailutils configuration file, `mail` reads the traditional '`mailrc`'-style configuration files. See Section 3.5.8 [Mail Configuration Files], page 89, for a detailed description of their format.

3.5.1 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.

`Mail` understands the following command line options:

`-A` *file*
`--attach=`*file*
: Attach *file* to the composed message. The encoding, content type, and content description are controlled by the `--encoding`, `--content-type`, and `--content-name` options, correspondingly.

  The option `--attach=-` instructs `mail` to read the file to be attached from the standard input. Interactive shell is disabled in this case.

`--attach-fd=`*fd*
: Read attachment body from the file descriptor *fd*. The descriptor must be open for reading. This option is useful when calling `mail` from another program.

  See the options `--encoding`, `--content-type`, `--content-name`, and `--content-filename`.

`-a` *header*`:` *value*
`--append=`*header*`:` *value*
: Append the given header to the composed message.

`--content-type=`*type*
: This options sets the content type to be used by all subsequent `--attach` options.

`--content-filename=`*name*
: Set the '`filename`' parameter in the '`Content-Disposition`' header for the next `--attach-fd` option.

`--content-name=`*text*
: Set the '`name`' parameter (description) in the '`Content-Type`' header for the next `--attach` or `--attach-fd` option.

`-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 3.5.8 [Mail Configuration Files], page 89), but before opening the mailbox.

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

`--encoding=`*enc*
: Sets content transfer encoding for use by the subsequent `--attach` options.

`-F`
`--byname`
: Record outgoing messages in a file named after the first recipient. The name is the login-name portion of the address found first on the '`To:`' line in the mail header.

`-f`
`--file`
: Operate on the mailbox given by the first non-optional command line argument. If there is no such argument, read messages from the user's `mbox` file. See Section 3.5.2 [Reading Mail], page 52, for more details about using this option.

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

`-i`
`--ignore`
: Ignore interrupts when composing the message.

`-M`
`--mime`
`--no-mime`
: The `--mime` option instructs `mail` to compose MIME messages. It is equivalent for `-E 'set mime'`, except that it is processed after all other options. The `--no-mime` disables the MIME compose mode, and is a shortcut for `-E 'set nomime'`,

`-N`
`--nosum`
: Do not display initial header summary.

`-n`
`--norc` Do not read the system-wide mailrc file. See Section 3.5.8 [Mail Configuration Files], page 89.

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

```
print *
quit
```

except that `mail --print` does not change status of the messages.

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

`-r` *address*
`--return-address=`*address*
Sets the return email address for outgoing mail. See [return-address], page 87.

`--skip-empty-attachments`
`--no-skip-empty-attachments`
Don't create attachments that would have zero-size body. This option affects all attachments created by `--attach` and `--attach-fd` options appearing after it in the command line, as well as the body of the original message.

To cancel its effect, use the `--no-skip-empty-attachments` option.

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

`-t`
`--to` Read recipients from the message header. Ignore addresses listed in the command line.

`-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).

The program also understands the common mailutils options (see Section 3.1.2 [Common Options], page 8.

3.5.2 Reading Mail

The `mail` utility operates on three kinds of mailboxes. The *user system mailbox* is the mailbox where the incoming mail for the user is stored. Its location is system-dependent and is determined using the common mailutils rules (see Section 3.2.8 [mailbox statement], page 19). The *personal mailbox* (or *mbox*, for short) is the default location for saving messages that have been read. By default it is `$HOME/mbox` or whatever file specified by the `MBOX` environment variable. Any other mailboxes are called *secondary mailboxes*.

When called without arguments, `mail` opens the system mailbox for the invoking user. The `--file` (`-f`) used without arguments instructs `mail` to operate on the personal mailbox instead. When this option and a single command line argument are used together, `mail` treats the argument as the pathname of the secondary mailbox to operate upon.

Notice that this argument is not an argument to the `--file` (`-f`) option itself, but rather the first non-optional argument on the command line. This means that any number of additional options are allowed between the `--file` option and the mailbox file name. For example, the following three invocations are equivalent:

```
$ mail -fin mymbox
$ mail -f mymbox -in
$ mail --file -in mymbox
$ mail --file -i mymbox -n
```

Additionally, for conformance to the GNU standards, the following form is also accepted:

```
$ mail --file=mymbox -i -n
```

The `--user` (`-u`) option allows the system administrator to assume another user identity for operating on this user's mailboxes. Obviously, it is available only to system administrators. For example:

```
mail --user=tom
```

reads mail from the system mailbox of the user '`tom`', and

```
mail --user=tom --file
```

reads mail from the personal mailbox of this user.

Unless you have started mail with `--norc` command line option, it will read the contents of the system-wide configuration file. Then it will read the contents of user configuration file, if it exists. For detailed description of these files, see Section 3.5.8 [Mail Configuration Files], page 89. After this initial setup, `mail` displays the first page of header lines (unless the `-N` option has been given), followed by a prompt, indicating that it is waiting for regular commands. Upon receiving a command, `mail` parses and executes it, displays the result on the screen, prints the prompt and waits for the next command. This process is continued until `mail` receives any of the commands '`quit`', '`exit`' or the end-of-file character (ASCII 4, or *C-D*).

Each message in the mailbox has a state that affects how it is retained or deleted upon closing the mailbox when terminating the program (see [the quit command], page 56) or when switching to another mailbox (see [the file command], page 57). The state is reflected in the header listing and can be changed during the session. The states are:

new The message is present in the system mailbox and has not been read by the user or moved to any other state. When `mail` terminates, messages in this state are retained in the system mailbox. If the mailbox is closed, such messages are moved to the '`unread`' state.

unread The message has been present in the system mailbox for more than one invocation of `mail` and has not been read by the user or moved to any other state. When `mail` terminates, messages in this state are retained in the system mailbox.

read The message has been read by the user, i.e. processed by one of the following commands: `copy`, `mbox`, `next`, `pipe`, `prev`, `print`, `Print`, `struct`, `top`, `type`, `Type`, `undelete`, or any of the following escapes (in message compose mode): `~f`, `~m`, `~F`, `~M`.

When `mail` terminates, messages in this state are handled depending on the mailbox they are in.

If `mail` was operating on the user system mailbox, all messages in state '`read`' are preserved. The location where they are preserved is determined by the `hold` variable (see Section 3.5.7 [Mail Variables], page 76). If it is not set (the default), the messages are moved to the user's `mbox`. If `hold` is set, the messages are held in the system mailbox instead.

The '`read`' messages in any other mailbox will be retained in their current location.

deleted The message has been processed by one of the following commands: '`delete`', '`dp`', '`dt`'. Messages in this state are ignored by any command, excepting '`undelete`', which changes their state back to '`read`'. When closing the mailbox, deleted messages are permanently removed from the mailbox.

preserved The message has been processed by the `preserve` (`hold`) command. When closing the mailbox, such messages are retained in the mailbox.

saved The message has been processed by one of the following commands: `save`, `write`. When `mail` terminates, messages in this state are handled depending on the mailbox they are in.

If `mail` was operating on the user system mailbox, the default behavior for '`saved`' messages is to remove them from the system mailbox. If, however, the `keepsave` variable is set, they are preserved using the same rules as for '`read`' messages (see above).

Saved messages in non-system mailboxes are retained in their current location.

Unless the mailbox is empty, exactly one of its messages will be marked as *current message*. Upon startup, current message is set to the first new message, if there is any, or the first unread message if there is any, or to the first message in the mailbox. In the header listing, current message is marked with the '`>`' sign at the beginning of the line. Current message is changed by any of the following commands: '`dp`', '`prev`', '`next`'.

3.5.2.1 Syntax of mail internal commands

Commands have the following syntax:

```
command [msglist] [argument ...]
```

A command is terminated by a newline character. Empty command (i.e. a newline character alone) is equivalent to '`next`' (see Section 3.5.2.4 [Moving Within a Mailbox], page 57).

In the syntax above, *command* is the command verb. Each command has long and short (abbreviated) form. Each of them can be used to invoke the command.

Many mail commands take a list of messages (*msglist*) to operate upon, which defaults to current message.

The list of messages 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

n (integer number) This specifier addresses the message with the given ordinal number in the mailbox.

n-*m* All messages with ordinal numbers between *n* and *m*, inclusive.

:*t* All messages of type *t*, where *t* can be any of:

'`d`' Deleted messages.

'`n`' New messages.

'`o`' Old messages (any message not in state '`read`' or '`new`').

'`r`' Messages in state '`read`'.

'`u`' Messages in state '`unread`'.

'`t`' Selects all tagged messages.

'`T`' Selects all untagged messages.

'`s`' Selects all messages in state '`saved`'.

[*header*:]/*string*[/]
Header match.

Selects all messages that contain header field *header* matching given *string*. If the variable `regex` is set, the *string* is assumed to be a POSIX regexp. (All comparison is case-insensitive in either case).

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

:/*string*[/] Message body match.

Selects all messages with body matching the *string*. The matching rules are the same as described above.

Some `mail` commands can operate on parts of multipart messages. In arguments to such commands, a message part is indicated by suffixing the message specifier by a dot and the number of that part in the message (message parts, as well as messages, are indexed from 1). For example, the command

```
write 4.1
```

will operate on part 1 of the message 4. Message parts can in turn be multipart messages. Such nested subparts are indicated in a similar manner, for example:

```
write 4.1.2
```

which means subpart 2 of the part 1 of message 4.[1]

The following are the examples of valid message lists:

3 Third message.

1-4 10 Messages from 1 through 4 and message 10.

4-* All messages starting from message 4.

/watch All messages with the word '`watch`' in the subject.

/watch :/watch
All messages with the word '`watch`' in the subject or body.

/watch :/watch $
Same as above plus the last message in the mailbox.

10.2 Part 2 of the multipart message 10.

3.5.2.2 Quitting the Program

Following commands quit the program:

`quit` [Mail command]
Terminates the session. The messages, marked with `delete` are removed. The messages in state '`read`' and '`saved`' are processed depending on the mailbox they are in.

If `mail` was operating on the user system mailbox, all messages in state '`read`' are preserved. The location where they are preserved is determined by the `hold` variable. If it is not set (the default), the messages are moved to the user's `mbox`. If `hold` is set, the messages are held in the system mailbox instead.

The default behavior for '`saved`' messages is to remove them from the system mailbox. If, however, the `keepsave` variable is set, they are preserved using the same rules as for '`read`' messages.

For non-system mailboxes, both '`read`' and '`saved`' messages are retained in their current location.

The same rules are followed when the mailbox is switched using the `file` command.

The program exits to the shell, unless saving the mailbox fails, in which case user can escape with the exit command.

`exit` [Mail command]
`ex` [Mail command]
`xit` [Mail command]
Program exits to the shell without modifying the mailbox it operates upon.

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

[1] For compatibility with earlier versions of `mailutils`, a message part can also be specified as a list of numbers enclosed in a pair of brackets, such as '`4[2]`', instead of '`4.2`'. This syntax is deprecated and will be withdrawn from future releases.

3.5.2.3 Obtaining Online Help

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

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

`list` [Mail command]
`*` [Mail command]
: Print a list of available commands.

`version` [Mail command]
`ve` [Mail command]
: Display program version.

`warranty` [Mail command]
`wa` [Mail command]
: Display program warranty statement.

3.5.2.4 Moving Within a Mailbox

`^` [Mail command]
: Move to the first undeleted message.

`$` [Mail command]
: Move to the last undeleted message.

`next` [Mail command]
`n` [Mail command]
: Move to the next message.

`previous` [Mail command]
`prev` [Mail command]
: Move to the previous message.

3.5.2.5 Changing Mailbox/Directory

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

`file` [*mailbox*] [Mail command]
`fi` [*mailbox*] [Mail command]
`folder` [*mailbox*] [Mail command]
`fold` [*mailbox*] [Mail command]
: When used without argument, prints the information about the current mailbox: the mailbox file name (or URL), total number of messages and the number of unread messages, e.g.:

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

Otherwise, closes the current mailbox and opens the mailbox named by the *mailbox* argument. When closing the current mailbox, its messages are processed according to their state (see [mail message states], page 53).

The *mailbox* argument can be the name of the existing file, a mailbox URL (see Chapter 2 [Mailbox], page 3), or any of the following shortcuts:

%	The system mailbox for the invoking user.
%*user*	The system mailbox for *user*.
#	The previous file.
&	The user's personal mailbox.
@	Secondary mailbox, given using the `-f` command line option.
+*file*	The named *file* in the folder directory. See [folder variable], page 79.

3.5.2.6 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*] [Mail command]
`di` [*header-field-list*] [Mail command]
`ignore` [*header-field-list*] [Mail command]
`ig` [*header-field-list*] [Mail command]

Add *header-field-list* to the ignored list. When used without arguments, this command prints the contents of ignored list.

`retain` [*header-field-list*] [Mail command]
`ret` [*header-field-list*] [Mail command]

Add *header-field-list* to the retained list. When used without arguments, this command prints the contents of retained list.

3.5.2.7 Displaying Information

`=` [Mail command]

Displays the current message number.

`headers` [*msglist*] [Mail command]
`h` [*msglist*] [Mail command]

Lists the current pageful of headers.

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

`z` [*arg*] [Mail command]
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*] [Mail command]
`si` [*msglist*] [Mail command]
Lists the message number and message size in bytes for each message in *msglist*.

`folders` [Mail command]
Displays the value of `folder` variable.

`summary` [Mail command]
`su` [Mail command]
Displays current mailbox summary. E.g.:

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

3.5.2.8 Displaying Messages

`print` [*msglist*] [Mail command]
`p` [*msglist*] [Mail command]
`type` [*msglist*] [Mail command]
`t` [*msglist*] [Mail command]
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*] [Mail command]
`P` [*msglist*] [Mail command]
`Type` [*msglist*] [Mail command]
`T` [*msglist*] [Mail command]
Like print but also prints out ignored header fields.

`decode` [*msglist*] [Mail command]
`dec` [*msglist*] [Mail command]
Print a multipart message. The `decode` command decodes and prints out specified messages. The *msglist* can contain message parts (see [message part specification], page 55), in which case only specified message parts will be output.

```
? decode 15.2
+--------------------------------------
| Message=15.2
| Type=message/delivery-status
| encoding=7bit
+--------------------------------------
Content-Type: message/delivery-status
...
```

`top` [*msglist*] [Mail command]
`to` [*msglist*] [Mail command]
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*] [Mail command]
`|` [[*msglist*] *shell-command*] [Mail command]
Pipe the contents of specified messages through *shell-command*. Without arguments, pipe the current message through the command given by the '`cmd`' variable (which must be set).

`struct` [*msglist*] [Mail command]
Prints the MIME structure of each message from *msglist*. Empty *msglist* means current message.

Example:

```
? struct 2
2      multipart/mixed            14k
2.1    text/plain                 296
2.2    application/octet-stream    5k
2.3    text/x-diff                31k
```

3.5.2.9 Marking Messages

`tag` [*msglist*] [Mail command]
`ta` [*msglist*] [Mail command]
Tag messages. The tagged messages can be referred to in message list using '`:t`' notation.

`untag` [*msglist*] [Mail command]
`unt` [*msglist*] [Mail command]
Clear tags from specified messages. To untag all messages tagged so far type

```
? untag :t
```

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

`unread` [*msglist*] [Mail command]
Marks each message from the *msglist* as not having been read.

3.5.2.10 Disposing of Messages

`delete` [*msglist*] [Mail command]
`d` [*msglist*] [Mail command]
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*] [Mail command]
`u` [*msglist*] [Mail command]
Clear delete mark from the specified messages.

`dp` [*msglist*] [Mail command]
`dt` [*msglist*] [Mail command]
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.

3.5.2.11 Saving Messages

`save` [[*msglist*] *mailbox*] [Mail command]
`s` [[*msglist*] *mailbox*] [Mail command]
Takes a message list and a file or mailbox name and appends each message in turn to that file or mailbox. The syntax for *mailbox* is the same as for the `file` command (see [Mailbox shortcuts], page 58). The name of the mailbox and number of lines and characters appended to it is echoed on the terminal. When writing to file, the numbers represent exact number of lines and characters appended to the file. When *file* specifies a mailbox, these numbers may differ by the amount of lines/characters needed to represent message envelope for that specific mailbox type.

Each saved message is marked for deletion as if with `delete` command, unless the variable `keepsave` is set.

`Save` [*msglist*] [Mail command]
`S` [*msglist*] [Mail command]
Like `save`, but the file to append messages to is named after the sender of the first message in *msglist*. The file name is selected as described in [saving mail by name], page 67. 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*] [Mail command]
`w` [[*msglist*] *file*] [Mail command]

Similar to `save`, except that only message body (without the header) is saved. The *msglist* can contain message parts (see [message part specification], page 55), in which case only specified message parts will be saved.

`Write` [*msglist*] [Mail command]
`W` [*msglist*] [Mail command]

Similar to `Save`, except that only message body (without the header) is saved. The *msglist* can contain message parts (see [message part specification], page 55), in which case only specified message parts will be saved.

`mbox` [*msglist*] [Mail command]
`mb` [*msglist*] [Mail command]

Mark list of messages to be saved in the user's personal mailbox (see Section 3.5.2 [Reading Mail], page 52) upon exiting via `quit` command. This is the default action for all read messages, unless you have variable `hold` set.

`touch` [*msglist*] [Mail command]
`tou` [*msglist*] [Mail command]

Touch the specified messages. If any message in *msglist* is not specifically deleted nor saved in a file, upon normal termination it will be acted upon as if it had been read (see [mail message states], page 53).

`copy` [[*msglist*] *file*] [Mail command]
`c` [[*msglist*] *file*] [Mail command]

Similar to `save`, except that saved messages are not marked as saved.

`Copy` [*msglist*] [Mail command]
`C` [*msglist*] [Mail command]

Similar to `Save`, except that saved messages are not marked as saved.

3.5.2.12 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*] [Mail command]
`e` [*msglist*] [Mail command]
Edits each message in *msglist* with the editor, specified in `EDITOR` environment variable.

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

3.5.2.13 Aliasing

`alias` [*alias* [*address...*]] [Mail command]
`a` [*alias* [*address...*]] [Mail command]
`group` [*alias* [*address...*]] [Mail command]
`g` [*alias* [*address...*]] [Mail command]
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...*] [Mail command]
`una` [*alias...*] [Mail command]
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...* [Mail command]
`alt` *name...* [Mail command]
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.

3.5.2.14 Replying

`mail` [*address...*] [Mail command]
`m` [*address...*] [Mail command]
Switches to compose mode. After composing the message, sends it to the specified addresses.

If the `record` variable is set, the composed message will be saved in the folder named by it.

`Mail` [*address...*] [Mail command]
`M` [*address...*] [Mail command]
Same as `mail`, but the name of the file to save the composed message is derived from its first recipient as outlined below.

If the `outfolder` variable is set, and has a string value *s*, the filename is *`s/recipient`*. If it is a boolean, then the `folder` variable is consulted. If it is set, then the filename is *`folder/recipient`*. Otherwise, the message will not be saved.

The value *recipient* is derived from the email of the first recipient of the message. By default it is a local part of that email. If the `outfilename` variable has the value '`domain`', the domain part of the email is used. If this variable is set to '`email`', then entire email address is used.

See [saving mail by name], page 67, for a detailed discussion.

`reply` [*message*] [Mail command]
`respond` [*message*] [Mail command]
`r` [*message*] [Mail command]

Mail a reply message to all recipients included in the header of the *message*. The subject header is formed by concatenating the value of the `replyprefix` variable and the subject from the message. If `record` is set to a filename, the response is saved at the end of that file.

`Reply` [*msglist*] [Mail command]
`Respond` [*msglist*] [Mail command]
`R` [*msglist*] [Mail command]

Mail a reply message to the sender of each message in the *msglist*. The subject header is formed by concatenating the value of the `replyprefix` variable and the subject header of from the first message in *msglist*. If `record` is set to a filename, the response is saved at the end of that file.

Notice, that setting mail variable `flipr` (see Section 3.5.7 [Mail Variables], page 76) swaps the meanings of the two above commands

`followup` [*message*] [Mail command]
`fo` [*message*] [Mail command]

Respond to *message*, recording the response in a file whose name is derived from the author of the message. See [saving mail by name], page 67, for a discussion of how the file name is selected.

`Followup` [*msglist*] [Mail command]
`F` [*msglist*] [Mail command]

Same as `Reply`, but the response is saved in a file whose name is derived from the author of the first message. See [saving mail by name], page 67, for a detailed discussion of how the file name is selected.

By default, `mail` will preserve personal email parts when forming lists of recipient addresses. If this is not desired, unset the `fullnames` variable (see [fullnames], page 79).

To determine the sender of the message `mail` uses the list of sender fields (see Section 3.5.2.15 [Controlling Sender Fields], page 65). 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
```

3.5.2.15 Controlling Sender Fields

When `mail` needs to know the sender of a message, it looks it up in one or more headers of that message. Such headers constitute a *sender list*. The first header from the list that is present in the message and has a non-empty value is used. If none is found or if the sender list is empty, the value of the message envelope is used.

The commands `sender` and `nosender` manipulate the sender 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
```

3.5.2.16 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.

3.5.2.17 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 `!`.

3.5.3 Saving and Recording

Several commands discussed in the previous section save messages in a disk file. The name of that file is either obtained from the `record` variable (*recording mail*) or is derived from the first recipient of the message (*saving by name*).

The following commands record mails:

- `mail`
- `reply`
- `Reply`

The following commands save mail by name:

- `Copy`
- `Save`
- `Mail`
- `followup`
- `Followup`

Saving mail by name is controlled by three mail variables: `outfolder`, `folder`, and `outfilename`. The first, `outfolder`, is a boolean variable which, when set, enables saving mail by name. The `folder` variable defines a directory where mail files are stored. Name of file in that directory where the message will be saved is derived from the message recipient[2]. This process is controlled by the `outfilename` variable: if its value is '`local`', the file is named by the local part of the email (this is the default). If it is '`domain`', the domain part is used instead. Finally, if it's value is '`email`', the entire email is used.

As a GNU extension, `outfolder` can be a string variable. In that case its value names the directory to use instead of `folder`.

The `mailx` variable, if set, disables GNU extensions. In this case, `outfolder` is used as a boolean value, and file names are derived from the local part of the email, ignoring the `outfilename` value.

3.5.4 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 and the standard input is connected to a terminal. 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 3.5.7 [Mail Variables], page 76).

3.5.4.1 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.

[2] In case of `Copy` and `Save`, message sender is used instead

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.

3.5.4.2 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.

3.5.4.3 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.

3.5.4.4 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.

3.5.4.5 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.

3.5.4.6 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
```

3.5.4.7 Attaching a File to the Message

The '`~+`' escape attaches a file to the message. It takes one to three arguments. The first argument supplies the name of the file to attach:

```
~+ myfile.txt
```

The file will be attached with default content-type '`application/octet-stream`', and encoding '`base64`' (these can be altered by the `--content-type` and `--encoding` command line options, correspondingly).

Optional second argument defines the content type to be used instead of the default one. Optional third argument defines the encoding, e.g.:

```
~+ myfile.html text/html base64
```

To list the files attached so far, use the '`~l`' escape:

```
~l
multipart/mixed
   1 myfile.html text/html base64
```

The first line of the output shows the content type of the message. Each subsequent line contains the ordinal number of the attachment, the name of the file, content-type and transfer encoding used.

The '`~/`' escape toggles the content type bewteen '`multipart/mixed`', and '`multipart/alternative`'. The new value of the content type is displayed on the screen.

The '`~^`' escape removes attachments. Its argument is the number of the attachment to remove, e.g.:

```
~^ 1
```

3.5.4.8 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.

3.5.4.9 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.

3.5.4.10 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.

3.5.4.11 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.

3.5.4.12 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`.

3.5.4.13 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.

3.5.5 Composing Multipart Messages

Multipart messages (or MIME, for short) can be used to send text in character sets other than ASCII, attach non-text files, send multiple parts in alternative formats, etc.

Technically speaking, the boolean variable `mime` controls this feature. If it is set (see [Setting and Unsetting the Variables], page 74), `MIME` will create MIME messages by default. The variable can be set in the global or user configuration file (see Section 3.5.8 [Mail Configuration Files], page 89), using the following command:

```
set mime
```

It can also be set from the command line, using the `--mime` option.

GNU `mail` automatically turns on the MIME mode, when it is requested to send a non-plaintext message, or a message in character set other than ASCII, when the encoding is specified, or when attachments are given.

To send a message in another character set, specify it with the `--content-type` option:

```
mail --content-type 'text/plain; charset=utf-8'
```

The `--encoding` specifies the encoding to use:

```
mail --content-type 'text/plain; charset=utf-8' --encoding=base64
```

Its argument is any encoding supported by GNU mailutils. The two most often used encodings are '`base64`' and '`quoted-printable`'.

To specify the charset from `mail` interactive section, enable the "edit headers" mode (`set editheaders`) and add the needed `Content-Type` header manually.

GNU `mail` also gives you a possibility to attach files to the message being sent.

The simplest way to attach a file from command line is by using the `--attach` (`-A`) option. Its argument specifies the file to attach. For example, the following will attach the content of the file `archive.tar`:

```
$ mail --attach=archive.tar
```

By default, the content type will be set to 'application/octet-stream', and the attachment will be encoded using the 'base64' encoding. To change the content type, use the --content-type option. For example, to send an HTML attachment:

```
$ mail --content-type=text/html --attach=in.html
```

The --content-type option affects all --attach options that follow it, and the message body (if any). To change the content type, simply add another --content-type option. For example, to send both the HTML file and the archive:

```
$ mail --content-type=text/html --attach=in.html \
       --content-type=application/x-tar --attach=archive.tar
```

To change the content type of the message body when sending a message with attachments, use the trailing --content-type option, i.e. the option not followed by another --attach option:

```
$ mail --content-type=text/html --attach=in.html \
       --content-type=application/x-tar --attach=archive.tar \
       --content-type=text/plain
```

This example adds two attachments with different content types and switched back to the 'text/plain' content type for the message body.

The encoding to use is set up by the --encoding option. As well as --content-type, this option affects all attachments supplied after it in the command line as well as the message body read from the standard input, until changed by the eventual next instance of the same option. Extending the above example:

```
$ mail --content-type=text/html --encoding=quoted-printable \
       --attach=in.html \
       --content-type=application/x-tar --encoding=base64 \
       --attach=archive.tar
```

A trailing --encoding option sets the encoding of the message body.

Each attachment can also be assigned a *description* and a *file name*. Normally, these are the same as the file name supplied with the --attach option. However, you can change either or both of them using the --content-name and --content-filename, correspondingly. Both of these options affect only the next --attach (or --attach-fd, see below) option.

By default, the message will be assigned the content type 'multipart/mixed'. To change it to 'multipart/alternative', use the --alternative command line option. Using this option also sets the 'Content-Disposition' header of each attached message to 'inline'.

All the examples above will enter the usual interactive shell, allowing you to compose the body of the message. If that's not needed, the non-interactive use can be forced by redirecting /dev/null to the standard input, e.g.:

```
$ mail --attach=archive.tar < /dev/null
```

This will normally produce a message saying:

```
mail: Null message body; hope that's ok
```

To suppress this message, unset the 'nullbodymsg' variable, as shown in the example below:

```
$ mail -E 'set nonullbodymsg' --attach=archive.tar < /dev/null
```

The option `--attach=-` forces `mail` to read the file to be attached from the standard input stream. This option disables the interactive mode and sets '`nonullbodymsg`' implicitly, so that the above example can be rewritten as:

```
$ mail --attach=- < archive.tar
```

Special option is provided to facilitate the use of `mail` in scripts. The `--attach-fd=`*N* instructs the program to read the data to be attached from the file descriptor *N*. The above example is equivalent to:

```
$ mail --attach-fd=0 < archive.tar
```

Attachments created with this option have neither filename nor description set, so normally the use of `--content-name` and/or `--content-filename` is advised.

The option `--skip-empty-attachments` instructs `mail` to skip creating attachments that would have zero-size body. This option affects all attachments created by `--attach` and `--attach-fd` options appearing after it in the command line. It also affects the handling of the original message body. To cancel its effect, use the `--no-skip-empty-attachments` option.

Here are some examples illustrating how it works.

First, consider the following command line

```
$ mail --attach=archive.tar </dev/null
```

Assume that `archive.tar` is not empty.

This will create a MIME message of two parts: the first part having '`text/html`' type and empty body, and the second part of type '`application/octet-stream`', with the content copied from the file `archive.tar`.

Now, if you do:

```
$ mail --attach=archive.tar --skip-empty-attachments </dev/null
```

then the created MIME message will contain only one part: that containing `archive.tar`.

If the file `archive.tar` has zero length, the resulting archive will still contain the '`application/octet-stream`' part of zero length. However, if you place the `--skip-empty-attachments` option before `--attach`, then the produced message will be empty.

The following Perl program serves as an example of using `mail` from a script to construct a MIME message on the fly. It scans all mounted file systems for executable files that have setuid or setgid bits set and reports the names of those files in separate attachments. Each attachment is named after the mountpoint it describes.

The script begins with the usual prologue stating the modules that will be used:

```
#!/usr/bin/perl

use strict;
use autodie;
```

Then global variables are declared. The '`@rcpt`' array contains the email addresses of the recipients:

```
my @rcpt= 'root@example.com';
```

The '`@cmd`' variable holds the `mail` command line. It will be augmented for each file system. The initial value is set as follows:

```
my @cmd = ('mail',
```

```
            '-E set nonullbodymsg',
            '--content-type=text/plain');
```

The `find` utility will be used to locate the files. The script will start as many instances as there are mountpoints. Those instances will be run in parallel and their standard output streams will be connected to file descriptors passed to `mail` invocation in `--attach-fd` options.

The descriptors will be held in '`@fds`' array. This will prevent them from being wiped out by the garbage collector. Furthermore, care should be taken to ensure that the `O_CLOEXEC` flag be not set for these descriptors. This sample script takes a simplistic approach: it instructs Perl not to close first 255 descriptors when executing another programs:

```
my @fds;
$^F = 255;
```

The following code obtains the list of mount points:

```
open(my $in, '-|', 'mount -t nonfs,noproc,nosysfs,notmpfs');
while (<$in>) {
    chomp;
    if (/^\S+ on (?<mpoint>.+) type (?<fstype>.+) /) {
```

For each mountpoint, the `find` command line is constructed and launched. The file descriptor is pushed to the '`@fds`' array to prevent it from being collected by the garbage collector:

```
open(my $fd, '-|',
     "find ${mpoint} -xdev -type f"
             . " \\( -perm -u+x -o -perm -g+x -o -perm -o+x \\)"
             . " \\( -perm -u+s -o -perm -g+s \\) -print");
push @fds, $fd;
```

Now, the `mail` command is instructed to create next attachment from that file descriptor:

```
        my $mpname = ${mpoint};
        $mpname =~ tr{/}{%};
        push @cmd,
             "--content-name=Set[ug]id files on ${mpoint} (type ${fstype})",
             "--content-filename=$mpname.list",
             '--attach-fd=' . fileno($fd);
    }
}
close $in;
```

Finally, the emails of the recipients are added to the command line, the standard input is closed to make sure `mail` won't enter the interactive mode and the constructed command is executed:

```
push @cmd, @rcpt;
close STDIN;
system(@cmd);
```

3.5.6 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 are 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.

To undo the effect of the previous example, do:

```
? unset askcc crt indentprefix
```

When used without arguments, both `set` or `unset` list all currently defined variables. The form of this listing is controlled by `variable-pretty-print` (`varpp`) variable. If it is set, a description precedes each variable, e.g.:

```
# prompt user for subject before composing the message
ask
# prompt user for cc before composing the message
askcc
# output character set for decoded header fields
charset="auto"
# number of columns on terminal screen
columns=80
```

If `variable-pretty-print` is not set, only the settings are shown, e.g.:

```
ask
askcc
```

```
charset="auto"
columns=80
```

A special command is provided to list all internal `mail` variables:

```
variable [names...]
```

If used without arguments, it prints all known internal variables. If arguments are given, it displays only those internal variables that are listed in command line. For each variable, this command prints its name, data type, current value and a short description. For example:

```
? variable ask datefield
ask, asksub
Type: boolean
Current value: yes
prompt user for subject before composing the message

datefield
Type: boolean
Current value: [not set]
get date from the `Date:' header, instead of the envelope
```

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@example.com gray@example.org
set
```

3.5.7 How to Alter the Behavior of `mail`

Following variables control the behavior of GNU `mail`:

`boolean append` [mail]

Default: True
Comment: Read-Only

Messages saved in `mbox` are appended to the end, rather than prepended. This is the default and cannot be changed. This variable exists only for compatibility with other `mailx` implementations.

`boolean appenddeadletter` [mail]

Default: False

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

`boolean askbcc` [mail]

Default: False

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

`boolean askcc` [mail]

Default: True

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

`boolean asksub` [mail]

Default: True in interactive mode, False otherwise.

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

`boolean autoinc` [mail]

Default: True

Automatically incorporate newly arrived messages.

`boolean autoprint` [mail]

Default: False

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

`boolean bang` [mail]

Default: False

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

See Section 3.5.4.13 [Executing Shell Commands], page 70, for details on the `!` escape.

`boolean datefield` [mail]

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 local time. Notice, that for messages lacking this field `mail` will fall back to using SMTP envelope.

See [fromfield], page 79.

`string charset` [mail]

Default: '`auto`'

The value of this variable is the character set used for input and output operations. If the value is '`auto`', `mail` will try to deduce the name of the character set from the value of `LC_ALL` environment variable. If the variable contains the character set part (e.g. '`nb_NO.utf-8`'), it will be used. Otherwise, `mail` will look up in its built-in database the value of the character for this language/territory combination. If `LC_ALL` is not set, the `LANG` environment variable is inspected.

The value of `charset` controls both input and output operations. On input, it is used to set the value of the '`charset`' parameter in the '`Content-Type`' MIME header, if its value begins with '`text/`' and the '`charset`' parameter is not present.

On output, it is used to display values of the header fields encodied using RFC 2047. If the variable is unset, no decoding is performed and the fields are printed as they are. Otherwise, they are recoded to that character set.

`string cmd` [mail]

Default: Unset

Contains default shell command for `pipe`.

`numeric columns` [mail]

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.

`numeric crt` [mail]
`boolean crt` [mail]

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.

`boolean debug` [mail]
`string debug` [mail]

Default: Unset

Sets mailutils debug level. If set to string, the value must be a valid Mailutils debugging specification. See [Debug Statement], page 18, for a description.

If unset (i.e. `set nodebug`), clears and disables all debugging information. If set to '`true`' (i.e. `set debug`), sets maximum debugging ('`<trace7`') on mailbox and its underlying objects.

`string decode-fallback` [mail]

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`'.

`boolean dot` [mail]

Default: False

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

`boolean editheaders` [mail]

Default: False

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

`boolean emptystart` [mail]

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.

`string escape` [mail]

Default: '`~`'

Current value of the command escape character.

`boolean flipr` [mail]

Default: Unset

If set, the variable `flipr` swaps the meanings of `reply` and `Reply` commands (see Section 3.5.2.14 [Replying], page 63).

`string folder` [mail]

Default: Unset

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

`boolean fromfield` [mail]

Default: True

By default the sender address is taken from the '`From`' header. Unsetting this variable tells `mail` to obtain it from the SMTP envelope instead.

See [datefield], page 77.

`boolean fullnames` [mail]

Default: True

Preserve personal parts (comments) of recipient addresses when replying to a message.

When unset, only emails will be used.

See Section 3.5.2.14 [Replying], page 63.

`boolean header` [mail]

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

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

`string headline` [mail]

Default: '`%>%a%4m %18f %16d %3l/%-5o %s`'

Format string to use for the header summary. The '`%`' character introduces a *format specifier*. The format specifier consists of optional alignment specifier ('`+`' or '`-`' sign), optional output width and the specifier letter. Format specifiers are replaced on output with the corresponding piece of information from the message being described.

The '`-`' character immediately following '`%`' indicates that this field should be left aligned. The '`+`' character indicates right alignment. Default alignment depends on the type of the specifier: the specifiers that produce numeric values ('`%l`', '`%m`', and '`%o`') are aligned to the right, whereas the ones producing string or date/time values are aligned to the left.

A number following '`%`' or the alignment flag, indicates the field width.

Consider the '`%m`' specifier as an example:

%m Print current message number. Take as much screen columns as necessary for output.

%4m
%+4m Print current message number. Use exactly 4 screen columns, truncating the output if it does not fit that width. Align the output to the right.

%-4m Same as above, but align to the left.

Valid format specifiers are:

%a Message attribute. One of the following letters, or a single horizontal space, if none of them applies:

'`M`'	the message was copied to the mailbox (`mbox` command)
'`P`'	the message was preserved (`hold` command)
'`*`'	the message was saved (`save` or `Save`)
'`T`'	the message was tagged (`tag`)
'`R`'	the message was read
'`N`'	the message is new (was not seen)
'`U`'	the message was seen, but wasn't read

%d The date when the message was received. It is determined from the message header defined by the '`datefield`' variable (see [datefield], page 77). If that variable is not set, or the requested header is not present in the message, the date from the envelope is used.

The output is formatted according to the following format specification (see Appendix C [Date/time Format String], page 213):

```
%a %b %e %H:%M
```

I.e.: abbreviated weekday name, abbreviated month name, day of the month as a decimal number, followed by hour and minutes. All names are displayed according to the current locale.

%D{*fmt*} Same as '`%d`', but the date is formatted according to the date/time format *fmt*. It is essentially a C '`strftime`' format string, described in detail in Appendix C [Date/time Format String], page 213.

For example:

```
set headline="%4m %20D{%Y-%m-%dT%H:%M:%S}"
```

Note, that the opening '`{`' must follow the format letter without any intervening whitespace. If *fmt* contains '`{`', '`}`', or '`\`', these characters must be escaped with backslash (e.g. '`\{`').

%D*f* A simplified form of the '`%D`' specifier. It is equivalent to

```
%D{%f}
```

where *f* is a single '`strftime`' specifier letter. It can be preceded by '`E`' or '`O`', if the Single UNIX Specification allows such usage (see [conversion specs], page 215), e.g. '`%DOU`'.

Notice, that '`%D`' not followed by a valid time format in either of the above forms is treated as unknown specifier.

%f The email address of the message sender.

%l The number of lines of the message.

%m Message number.

%o The number of octets (bytes) in the message.

%s Message subject (if any).

%S Message subject (if any) in double quotes.

%> A '`>`' for the current message, otherwise a space.

%< A '`<`' for the current message, otherwise a space.

%% A '`%`' character.

`boolean hold` [mail]

Default: False

Determines the location where to store the messages in state '`read`' and (if the `keepsave` is also set) '`saved`'. When set, these messages will be retained in the system mailbox.

When not set (the default), such messages will be stored in the user's personal mailbox.

See [read messages], page 53, and See [saved messages], page 54, for a detailed information on how such messages are processed when the mailbox is being closed.

See [keepsave], page 82, for the discussion of the `keepsave` variable.

`boolean ignore` [mail]

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.

`boolean ignoreeof` [mail]

Default: False

Controls whether typing EOF character terminates the letter being composed.

`string indentprefix` [mail]

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

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

`boolean inplacealiases` [mail]

Default: False

If set, `mail` will expand aliases in the address header field before entering send mode (see Section 3.5.4 [Composing Mail], page 67). By default, the address header fields are left intact while composing, the alias expansion takes place immediately before sending message.

`boolean keep` [mail]

Comment: Read-Only
Default: True

Truncate the user's system mailbox when it is empty, instead of removing it. This is the default and cannot be changed. This variable exists only for compatibility with other `mailx` implementations.

`boolean keepsave` [mail]

Default: False

Controls whether saved messages should be retained. The location where they will be retained is controlled by the `hold` variable (see [the hold variable], page 81).

This variable is in effect only when operating upon the user's system mailbox.

See [saved messages], page 54, for a detailed information on how the saved messages are processed when the mailbox is being closed.

`boolean mailx` [mail]

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.
- The `outfolder` variable is treated as boolean. see [outfolder], page 85.

- The value of `outfilename` is ignored (assumed to be '`local`'). see [outfilename], page 85.
- The values of `folder` and `record` variables are assumed relative to the home directory, unless they begin with '`/`', '`~`', or '`+`'.
- If the value of the `sendmail` variable does not begin with a scheme specification, '`sendmail:/`' is assumed. See [sendmail mail variable], page 87.

`boolean metamail` [mail]
`string metamail` [mail]

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"
```

`string mime` [mail]

Default: False

If set, this variable instructs `mail` to compose MIME messages.

It can be set from the command line using `--mime` option.

`string mimenoask` [mail]

Default: Unset

By default `mail` asks for confirmation 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`'.

`boolean metoo` [mail]

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.

`string mode` [mail]

Comment: Read-Only
Default: The name of current operation mode.

This variable keeps the name of the current operation mode. Its possible values are:

headers The program is started with the `--headers` (`-H`) command line option (see Section 3.5.1 [Invoking Mail], page 50).

exist The program is started with the `--exist` (`-e`) command line option (see Section 3.5.1 [Invoking Mail], page 50).

print The program is started with the `--print` (`-p`) command line option (see Section 3.5.1 [Invoking Mail], page 50).

read The program operates in read mode. This is the default.

send The program operates in send mode. This means it was given one or more recipient addresses in the command line.

`boolean nullbody` [mail]

Default: True

Controls whether `mail` accepts messages with an empty body. The default value, `true`, means such messages are sent, and a warning (traditionally saying '`Null message body; hope that's ok`') is displayed. The text of the warning can be set using `nullbodymsg` variable (see below).

If `nullbody` is unset, `mail` will silently ignore such messages. This can be useful in `crontab` files, to avoid sending mails when nothing important happens. For example, the `crontab` entry below will send mail only if the utility `some-prog` outputs something on its standard output or error:

```
*/5 * * * * some-prog 2>&1 | \
   /bin/mail -E'set nonullbody' -s 'Periodic synchronization'
```

`string nullbodymsg` [mail]

Default: '`Null message body; hope that's ok`'

Text of the warning displayed by `mail` before sending an empty message. When available, the translation of this text, in accordance with the current locale, is displayed.

Unsetting this variable disables the warning.

`boolean onehop` [mail]

Default: Unset

This variable is not used. It exists for compatibility with other `mailx` implementations and for future use.

`string outfilename` [mail]

Comment: Three-state: '`local`', '`email`', '`domain`'.
Default: '`local`'

Defines the algorithm to convert the recipient email to the name of the file used to record outgoing messages to that recipient. This affects the following commands: `Copy`, `Save`, `Mail`, `followup`, and `Followup`. The following values are allowed:

`local` Local part of the email address is taken as the file name. This is the default.

`email` Entire email is takes as the file name.

`domain` Domain part of the email is used as the file name.

`boolean outfolder` [mail]
`string outfolder` [mail]

Default: Unset

When set as boolean, causes the files used to record outgoing messages to be located in the directory specified by the `folder` variable (unless the pathname is absolute).

If set to a string value, names the directory where to store these files.

This variable affects the following commands: `Copy`, `Save`, `Mail`, `followup`, and `Followup`.

In mailx compatibility mode, only boolean value is allowed. see [mailx mail variable], page 82.

`boolean page` [mail]

Default: Unset

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

`string PID` [mail]

Comment: Read-Only
Default: PID of the process.

PID of the current `mail` process.

`string prompt` [mail]

Default: `"? "`

Contains the command prompt sequence.

`boolean quiet` [mail]

Default: Unset

This variable is not used. It exists for compatibility with other `mailx` implementations and for future use.

`boolean quit` [mail]

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

When set, causes keyboard interrupts to terminate the program.

`boolean rc` [mail]

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 3.5.8 [Mail Configuration Files], page 89.

`boolean readonly` [mail]

Default: False

When set, mailboxes are opened in readonly mode. In this mode, any `mail` commands that alter the contents of the mailbox are disabled. These commands include, but are not limited to: `delete`, `save` and `mbox`.

`string record` [mail]

Default: Unset

When set, outgoing messages produced by the following commmands will be saved to the named file: `mail`, `reply`, `Reply`.

See also [outfolder], page 85, and [outfilename], page 85.

`boolean recursivealiases` [mail]

Default: True

When set, `mail` will expand aliases recursively.

`boolean regex` [mail]

Default: True.

If set, enables the use of regular expressions in '`/.../`' message specifications.

`string replyprefix` [mail]

Default: '`Re: `'

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

`string replyregex` [mail]

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 value 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).

`string return-address` [mail]

Default: unset

Sets the return email address to use when sending messages. If unset, return address is composed from the current user name and the host name.

`boolean save` [mail]

Default: True.

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

`numeric screen` [mail]

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. See also [crt], page 78.

`string sendmail` [mail]

Default: '`sendmail:/usr/lib/sendmail`'

Contains URL of the mail transport agent. If the value begins with a scheme specifier, it must be one of the mailer URL schemes supported by mailutils (see [mailer URL], page 25). Otherwise, if not in mailx compatibility mode, the value starting with directory separator ('`/`') is treated as the external command that will be started as is and the composed message will be piped to its standard input.

In mailx compatibility mode (see [mailx mail variable], page 82), the '`sendmail:`' prefix is assumed.

`boolean sendwait` [mail]

Default: Unset

This variable is not used. It exists for compatibility with other `mailx` implementations and for future use.

`string Sign` [mail]

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.

`string sign` [mail]

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.

`boolean showenvelope` [mail]

Default: Unset

If this variable is set, the `print` command will include the SMTP envelope in its output.

`boolean showto` [mail]

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.

`string subject` [mail]

Default: Unset

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

`numeric toplines` [mail]

Default: 5

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

`boolean variable-pretty-print` [mail]
`boolean varpp` [mail]

Default: False

If this variable is set, the listing output by `set` contains short descriptions before each variable. See [Setting and Unsetting the Variables], page 74.

`boolean variable-strict` [mail]
`boolean varstrict` [mail]

Default: False

Setting this variable enables strict control over variable settings. In this mode, `mail` refuses to set read-only variables. Also, if the user is trying to set an unknown variable, `mail` prints a warning.

See [Setting and Unsetting the Variables], page 74.

`boolean verbose` [mail]

Default: False

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

`boolean useragent` [mail]

Default: True

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

```
User-Agent: mail (GNU Mailutils 3.19)
```

`boolean xmailer` [mail]

This header is retained for compatibility with previous releases of GNU Mailutils. Since version 3.13 it is an alias for `useragent`.

3.5.8 Personal and System-wide Configuration Files

`Mail` reads its configuration from several places. First, it reads the usual `Mailutils` configuration files, as described in Section 3.2 [configuration], page 9.

The following configuration file statements affect `mail` behavior:

Statement	Reference
address	See Section 3.2.6 [address statements], page 18.
debug	See Section 3.2.7 [debug statement], page 18.
mailbox	See Section 3.2.8 [mailbox statement], page 19.
locking	See Section 3.2.10 [locking statement], page 22.
mailer	See Section 3.2.11 [mailer statement], page 24.

After processing the usual Mailutils configuration files, `mail` reads two command files prescribed by POSIX: the system-wide command file, and the user's command file. Each line read from these files is processed like a usual `mail` command. As with mailutils configuration files, these files are not required to exist.

Per-user 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`.

Although this setup might seem unnecessarily complicated, it is based on sound reasons. First of all, since `mail` is a part of `GNU mailutils`, it is supposed to use the same settings as its remaining parts. On the other hand, the `mail` utility is part of POSIX standard, which requires it to read the second pair of files.

`Mail` command files normally consist of several `set` statements (see [Setting and Unsetting the Variables], page 74), modifying some variables, discussed in Section 3.5.7 [Mail Variables], page 76. For example:

```
set hold emptystart editheaders
retain from to subject reply-to in-reply-to date
alt user@example.org
```

Certain `mail` settings duplicate those of Mailutils. Here is a list of these:

Mail variable	Mailutils setting
`debug`	`.debug.level`
`folder`	`.mailbox.folder`
`sendmail`	`.mailer.url`

The rule of thumb that helps to decide which setting to use is simple: if the setting is common for all tools running on that server, place it in the site-wide Mailutils configuration file. Otherwise, place it in one of `mail` command files.

To trace reading and parsing the Mailutils configuration files, use `--config-lint` and `--config-verbose` options. To skip the user configuration file, use the `--no-user-config`. To skip the system-wide configuration file, use the `--no-site-config` option. To skip both, use `--no-config`. See Section 3.1.2 [Common Options], page 8, for details.

To skip reading the system-wide `mail` command file, use the `--norc` (`-N`) option. The per-user command file is `~/.mailrc` by default. You can instruct `mail` to read another file, by setting the environment variable `MAILRC` to the desired file name. To ignore per-user file altogether, set `MAILRC=/dev/null`.

3.6 `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 following configuration file statements affect the behaviour of `messages`:

Statement	**Reference**
debug	See Section 3.2.7 [debug statement], page 18.
tls	See Section 3.2.21 [tls statement], page 42.
mailbox	See Section 3.2.8 [mailbox statement], page 19.
locking	See Section 3.2.10 [locking statement], page 22.

In addition to the common mailutils options (see Section 3.1.2 [Common Options], page 8), the program accepts the following command line options:

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

3.7 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`.

The program was initially 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 fully backward-compatible with its Emacs predecessor, so it should run flawlessly with older versions of Emacs. Emacs versions starting from 22.1 contain improved `Rmail` interface and are able to take advantage of all new features mailutils `movemail` provides.

Apart from that use, `movemail` proved to be a useful tool for incorporating mail from remote mailboxes into the local one. See Fetching Mail with Movemail (`http://mailutils.org/wiki/Fetching_Mail_with_Movemail`), for a detailed discussion with usage recipes.

3.7.1 Movemail Configuration

The following configuration file statements affect the behavior of `movemail`:

`preserve` *bool* [Movemail Config]
: If *bool* is '`true`', do not remove messages from the source mailbox.

`reverse` *bool* [Movemail Config]
: If *bool* is '`true`', reverse message sorting order.

`emacs` *bool* [Movemail Config]
: If *bool* is '`true`', output information used by Emacs rmail interface.

`ignore-errors` *bool* [Movemail Config]
: Continue moving messages after errors. By default, `mailfromd` exits immediately if it cannot copy a message.

`program-id` *fmt* [Movemail Config]
: Set program identifier, i.e. a string which will prefix all diagnostic messages issued by the program. By default, program name is used.

  The *fmt* is a format string that may contain references to the following variables (see Section 3.2.2 [Variables], page 15):

  `progname` The program name.

  `source` URL of the source mailbox.

  `source_user`
  : User part of the source mailbox URL.

  `source_host`
  : Host part of the source mailbox URL.

  `source_path`
  : Path part of the source mailbox URL.

`dest` URL of the destination mailbox

`dest_user`
User part of the destination mailbox URL.

`dest_host`
Host part of the destination mailbox URL.

`dest_path`
Path part of the destination mailbox URL.

Setting `program-id` may be necessary if several `movemail` instances are run simultaneously (e.g. invoked from a script) to discern between the instances. For example:

```
program-id "${progname}: ${source} => ${dest}"
```

`uidl` *bool* [Movemail Config]
Avoid copying the message if a message with the same UIDL already exists in the destination mailbox.

`verbose` *level* [Movemail Config]
Set verbosity level.

`mailbox-ownership` *method-list* [Movemail Config]
Define list of methods for setting ownership of the destination mailbox. The *method-list* argument can contain the following elements:

copy-id Copy owner UID and GID from the source mailbox. This method works only with local mailboxes, i.e.: '`mbox`' (UNIX mailbox), '`maildir`' and '`mh`'.

copy-name
Get owner name from the source mailbox URL and obtain UID and GID for this user using mailutils authorization methods.

set-id=*uid*[:*gid*]
Set supplied *uid* and *gid*. If *gid* is not supplied, it is read from the `/etc/passwd` record for this UID.

set-name=*user*
Make destination mailbox owned by *user*.

`max-messages` *count* [Movemail Config]
Defines upper limit on the number of moved messages. Movemail will stop after transferring *count* messages.

By default, the number of messages is not limited.

`onerror` *actions* [Movemail Config]
Defines what to do if an error occurs when transferring a message. *actions* is a list of one or more of the following keywords:

abort Abort the transfer and terminate the program. This is the default action.

skip Skip to the next message.

delete Delete the affected message.

count Count this message as processed.

Each keyword can be prefixed with '`no`' to reverse its meaning.

The following standard Mailutils statements are supported:

Statement	Reference
debug	See Section 3.2.7 [debug statement], page 18.
tls	See Section 3.2.21 [tls statement], page 42.
mailbox	See Section 3.2.8 [mailbox statement], page 19.
locking	See Section 3.2.10 [locking statement], page 22.
pam	See Section 3.2.16 [pam statement], page 34.
sql	See Section 3.2.19 [sql statement], page 37.
virtdomain	See Section 3.2.17 [virtdomain statement], page 34.
radius	See Section 3.2.18 [radius statement], page 35.
ldap	See Section 3.2.20 [ldap statement], page 40.
auth	See Section 3.2.15 [auth statement], page 32.

3.7.2 Setting Destination Mailbox Ownership

==========

Editor's note:

The information in this node may be obsolete or otherwise inaccurate. This message will disappear, once this node revised.

==========

3.7.3 Movemail Usage Summary

`movemail [`*`option...`*`] `*`inbox destfile`*` [`*`password`*`]`

The first argument, *inbox*, is the url (see Chapter 2 [Mailbox], page 3) 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].

For compatibility with older implementations of `movemail`, the *source* argument can also have the form:

`po:`*`username`*`[:`*`pop-server`*`]`

where *pop-server* is the IP address or hostname of a POP3 server to connect to and *username* is the name of the user on that server. The password is then supplied by the third argument.

It is equivalent to the following URL:

`pop://`*`username`*`[:`*`password`*`]@`*`pop-server`*

In fact, whenever *source* refers to a remote mailbox, the *password* argument can be used to pass the password. However, the safer *ticket* method is of course preferred.

Options are one or more of the following:

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

[1] Rmail does not use this feature

`--ignore-errors`
: Try to continue after errors.

`--max-messages=`*count*
: Process at most *count* messages.

`--notify` Enable biff notification.

`--onerror=`*kw*`[,`*kw*`...]`
: What to do on errors. See [movemail-onerror], page 93, for a description of *kw*.

`-P` *modelist*
`--owner=`*modelist*
: Control mailbox ownership. *modelist* is a comma-separated list of one or more ownership change methods. See [mailbox-ownership-methods], page 93, for a description of available methods.

  This option is useful only when running `movemail` as root.

`-p`
`--preserve`
`--keep-messages`
: Don't remove transferred messages from the source mailbox.

`--program-id=`*fmt*
: Set program identifier for diagnostics (default: the program name). See [movemail-program-id], page 92, for a description of its argument.

`-r`
`--reverse`
: Reverse the order of retrieved messages.

`-u`
`--uidl` Use UIDLs to avoid downloading the same message twice.

`-v`
`--verbose`
: Increase verbosity level.

The common options are also understood (see Section 3.1.2 [Common Options], page 8).

3.8 `readmsg` — Extract Messages from a Folder

The `readmsg` utility extracts messages from a mailbox according to the criteria specified in the command line. These criteria are:

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 by default. This can changed using the `-i` (`--ignorecase`) option. Two more options are provided to control the matching algorithm: the `-g` (`--glob`) option instructs `readmsg` to treat arguments as shell globbing patterns and the `-r` (`--regex`) option instructs it to treat them as POSIX extended regular expressions. Needless to say, when using any of the two latter options, you should pay attention to escape the matching pattern to prevent it from being interpreted by the shell. E.g.:

   ```
   readmsg --regex 'staff.*meeting'
   ```

Unless requested otherwise, only the first message that matches the pattern is printed.

At least one command line argument or one informational option must be present in `readmsg` invocation. Informational options are: `--help` (`-?`), `--usage`, and `--version` (`-V`).

3.8.1 Invocation of `readmsg`.

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

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

`-e`
`--exact`
: Look for messages containing exactly the words given as arguments. This is the default. Other options changing this behavior are: `--glob`, `--regex`, and `--ignorecase`.

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

`-g`
`--glob`
: Treat non-option arguments as shell globbing patterns. For example, to select the first message with words '`morning`' and '`coffee`' with anything bewteen them:

```
readmsg --glob 'morning*coffee'
```

(notice quoting, which prevents the shell from interpreting the '`*`' prematurely).

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

`-i`
`--ignorecase`
Ignore the case of letters when looking for matching messages. E.g.:

```
readmsg --glob --ignorecase 'morning*coffee'
```

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

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

`-r`
`--regex` Treat non-option arguments as POSIX extended regular expressions.

`-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-"`.

See also Section 3.1.2 [Common Options], page 8.

3.8.2 Configuration of `readmsg`.

Following configuration statements affect the behavior of `readmsg`:

`header` *bool* [Readmsg Conf]
If *bool* is '`true`', display entire headers.

`weedlist` *str* [Readmsg Conf]
Set the weedlist. The *str* argument is a string, containing a list of header names, separated by whitespace, commands or colons. This corresponds to the `--weedlist` command line option (see Section 3.8.1 [Opt-readmsg], page 96).

`no-header` *bool* [Readmsg Conf]
If *bool* is '`true`', exclude all headers.

`form-feeds` *bool* [Readmsg Conf]
If *bool* is '`true`', output formfeed character between messages.

`folder` *url* [Readmsg Conf]
Set the URL of the mailbox folder to read.

`show-all-match` *bool* [Readmsg Conf]
If *bool* is '`true`', print all messages matching pattern, not only the first.

Statement	Reference
debug	See [Debug Statement], page 18.
tls	See [TLS Statement], page 42.
mailbox	See [Mailbox Statement], page 19.
locking	See [Locking Statement], page 22.

3.9 `decodemail` – Decode multipart messages

The `decodemail` utility is a filter program that reads messages from the input mailbox, decodes "textual" parts of each multipart message from a base64- or quoted-printable encoding to an 8-bit or 7-bit transfer encoding, and stores the processed messages in the output mailbox. All messages from the input mailbox are stored in the output, regardless of whether a change was made.

The message parts deemed to be textual are those whose '`Content-Type`' header matches a predefined, or user-defined, mime type pattern. In addition, encoded pieces of the '`From:`', '`To:`', '`Subject:`', etc., headers are decoded.

For example, `decodemail` makes this transformation:

```
Subject: =?utf-8?Q?The=20Baroque=20Enquirer=20|=20July=202020?=
⇒ Subject: The Baroque Enquirer | July 2020
```

The built-in list of textual content type patterns is:

```
text/*
application/*shell
application/shellscript
*/x-csrc
*/x-csource
*/x-diff
*/x-patch
*/x-perl
*/x-php
*/x-python
*/x-sh
```

These strings are matched as shell globbing patterns (see Section "glob" in *glob(7) manual page*).

More patterns can be added to this list using the `mime.text-type` configuration statement. See Section 3.2.9 [mime statement], page 21, for a detailed discussion, and the configuration section below for a simple example.

When processing old mesages you may encounter '`Content-Type`' headers whose value contains only type, but no subtype. To match such headers, use the pattern without '`/whatever`' part. E.g. '`text/*`' matches '`text/plain`' and '`text/html`', but does not match '`text`'. On the other hand, '`t*xt`' does not match '`text/plain`', but does match '`text`'.

Optionally, the decoded parts can be converted to another character set. By default, the character set is not changed.

3.9.1 Invocation of `decodemail`.

Usually, the utility is invoked as:

```
decodemail inbox outbox
```

where *inbox* and *outbox* are file names or URLs of the input and output mailboxes, correspondingly. The input mailbox is opened read-only and will not be modified in any way. In particular, the status of the processed messages will not change. If the output mailbox does not exist, it will be created. If it exists, the messages will be appended to it, preserving

any original messages that are already in it. This behavior can be changed using the `-t` (`--truncate`) option, described below.

The two mailboxes can be of different types. For example you can read input from an imap server and store it in local '`maildir`' box using the following command:

```
decodemail imap://user@example.com maildir:///var/mail/user
```

Both arguments can be omitted. If *outbox* is not supplied, the resulting mailbox will be printed on the standard output in Unix '`mbox`' format. If *inbox* is not supplied, the utility will open the system inbox for the current user and use it for input.

A consequence of these rules is that there is no simple way to read the input mailbox from standard input (the input must be seekable). If you need to do this, the normal procedure would be to save what would be standard input in a temporary file and then give that file as `decodemail`'s input.

The following command line options modify the `decodemail` behavior:

`-c, --charset=`*charset*
: Convert all textual parts from their original character set to the specified *charset*.

`-R, --recode`
: Convert all textual parts from their original character set to the current character set, as specified by the `LC_ALL` or `LANG` environment variable.

`--no-recode`
: Do not convert character sets. This is the default.

`-t, --truncate`
: If the output mailbox exists, truncate it before appending new messages.

`--no-truncate`
: Keep the existing messages in the output mailbox intact. This is the default.

Additionally, the Section 3.1.2 [Common Options], page 8, are also understood.

3.9.2 Configuration of `decodemail`.

The following common configuration statements affect the behavior of `decodemail`:

Statement	**Reference**
mime	See Section 3.2.9 [mime statement], page 21.
debug	See [Debug Statement], page 18.
mailbox	See [Mailbox Statement], page 19.
locking	See [Locking Statement], page 22.

Notably, the `mime` statement can be used to extend the list of types which are decoded. For example, in the file `~/.decodemail` (other locations are possible, see Section 3.2 [configuration], page 9), you could have:

```
# base64/qp decode these mime types also:
mime {
  text-type "application/x-bibtex";
  text-type "application/x-tex";
}
```

Since the list of textual mime types is open-ended, with new types being used at any time, we do not attempt to make the built-in list comprehensive.

3.9.3 Purpose and caveats of `decodemail`.

The principal use envisioned for this program is to decode messages in batch, after they are received.

Unfortunately, some mailers prefer to encode messages in their entirety in base64 (or quoted-printable), even when the content is entirely human-readable text. This makes straightforward use of `grep` or other standard commands impossible. The idea is for `decodemail` to rectify that, by making the message text readable again.

Besides personal mail, mailing list archives are another place where such decoding can be useful, as they are often searched with standard tools.

It is generally not recommended to run `decodemail` within a mail reader (which should be able to do the decoding itself), or directly in a terminal (since quite possibly there will be 8-bit output not in the current character set).

Although the output message from `decodemail` should be entirely equivalent to the input message, apart from the decoding, it is generally not identical. Because `decodemail` parses the input message and reconstructs it for output, there are usually small differences:

- In the envelope '`From `' line, multiple spaces are collapsed to one.
- A '`Content-Transfer-Encoding:`' header may be added where not previously present, or its value changed from '`8bit`' to '`7bit`', or vice versa. This may happen both for the message as a whole, and for a given mime part. `decodemail` looks at the actual content of the text and outputs '`Content-Transfer-Encoding:`' accordingly.
- A trailing space is inserted when a long header line is broken to occupy several lines (*header wrapping*).

```
SomeHeader:
  someextremelylongvaluethatcannotbebroken
```

- The non-tracing headers may be reordered, notably those that are mime-related.
- Any material before the first mime part of a mime multipart message is lost. By the standards, nothing should appear there. Typically if it does appear, it is a string such as '`This is a multi-part message in MIME format.`'.
- In mime parts, the charset specifications may no longer be quoted (if quoting is not necessary). For example, '`charset="utf-8"`' becomes '`charset=utf-8`'.
- The mime boundary strings will be changed.

If a discrepancy is created which actually affects message parsing or reading, that's most likely a bug, and please report it. Naturally, please send an exact input message to reproduce the problem.

3.10 `sieve`

==========

Editor's note:

The information in this node may be obsolete or otherwise inaccurate. This message will disappear, once this node revised.

==========

Sieve is a language for filtering e-mail messages at time of final delivery, described in RFC 3028. GNU Mailutils contains stand-alone *sieve interpreter*, which is described in detail below.

3.10.1 A Sieve Interpreter

The sieve interpreter `sieve` allows you to apply Sieve scripts to arbitrary number of mailboxes. GNU `sieve` implements a superset of the Sieve language as described in RFC 3028. See Chapter 5 [Sieve Language], page 171, for a description of the Sieve language. See Section 5.9 [GNU Extensions], page 195, for a discussion of differences between the GNU implementation of Sieve and its standard.

3.10.1.1 Invoking `sieve`

The `sieve` invocation syntax is:

```
sieve [options] script
```

Normally, *script* is the name of the disk file with the Sieve script. If *script* is a single dash, the script is read from the standard input. If the `-E` (`--expression`) option is given, *script* is taken to be the sieve script text.

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.

`--clear-library-path`
`--clearpath`
: Clear Sieve library path. See also Section 3.10.1.2 [Sieve Configuration], page 104.

`--clear-include-path`
: Clear Sieve include path. See also Section 3.10.1.2 [Sieve Configuration], page 104.

`-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 mailutils 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.

`--environment=`*name*`=`*value*
: Set sieve environment variable *name* to the *value*.

`-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. See also Section 3.10.1.2 [Sieve Configuration], page 104.

`-E`,
`--expression`
: Treat the *script* argument as Sieve program text.

`-I` *dir*
`--includedir=`*dir*
: Append directory *dir* to the list of directories searched for include files. See also Section 3.10.1.2 [Sieve Configuration], page 104.

`-f`
`--mbox-url=`*mbox*
: Mailbox to sieve (defaults to user's system mailbox). See also Section 3.10.1.2 [Sieve Configuration], page 104.

`-k`
`--keep-going`
: Keep on going if execution fails on a message. See also Section 3.10.1.2 [Sieve Configuration], page 104.

`-L` *dir*
`--libdir=`*dir*
: Append directory *dir* to the list of directories searched for library files. See also Section 3.10.1.2 [Sieve Configuration], page 104.

`--libdir-prefix=`*dir*
: Add *dir* to the beginning of the list of directories searched for library files.

`--line-info=`*bool*
: Print source location along with action logs (default).

`-M` *url*
`--mailer=`*url*
: Define the URL of the default mailer.

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

`--no-program-name`
: Do not prefix diagnostic messages with the program name.

`-t` *ticket*
`--ticket=`*ticket*
: Ticket file for mailbox authentication. See also Section 3.10.1.2 [Sieve Configuration], page 104.

`--variable=`*name*`=`*value*
: Set Sieve variable *name*. This option automatically inserts '`require "variables"`' at the top of the script.

`-v`
`--verbose`
: Log all actions executed. See also Section 3.10.1.2 [Sieve Configuration], page 104.

See also Section 3.1.2 [Common Options], page 8.

3.10.1.2 Sieve Configuration

The behavior of `sieve` is affected by the following configuration statements:

Statement	**Reference**
debug	See Section 3.2.7 [debug statement], page 18.
tls	See Section 3.2.21 [tls statement], page 42.
mailbox	See Section 3.2.8 [mailbox statement], page 19.
locking	See Section 3.2.10 [locking statement], page 22.
logging	See Section 3.2.5 [logging statement], page 17.
mailer	See Section 3.2.11 [mailer statement], page 24.

The following statements configure sieve-specific features:

`sieve { ... }` [Sieve Conf]
: This block statement configures search paths `sieve` uses to locate its loadable modules. See Section 5.4 [Require Statement], page 175, for a detailed information about loadable modules.

This statement may contain the following sub-statements:

`clear-library-path` *bool* [Sieve Conf]
: If *bool* is '`true`', clear library search path.

`clear-include-path` *bool* [Sieve Conf]
: If *bool* is '`true`', clear include search path.

`library-path` *path* [Sieve Conf]
: Add directories to `sieve` library search path. Argument is a string containing a colon-separated list of directories.

`library-path-prefix` *path* [Sieve Conf]
: Add directories to the beginning if the library search path. Argument is a string containing a colon-separated list of directories.

`include-path` *path* [Sieve Conf]
: Add directories to the include search path. Argument is a string containing a colon-separated list of directories.

`keep-going` *bool* [Sieve Conf]
: If *bool* is '`true`', do not abort if execution of a Sieve script fails on a particular message.

`mbox-url` *url* [Sieve Conf]
: Sets URL of the mailbox to be processed.

`ticket` *file* [Sieve Conf]
: Sets the name of the ticket file for user authentication.

`debug` *flags* [Sieve Conf]
: Sets Sieve debug flags. See Section 3.10.1.3 [Logging and Debugging], page 105, for a detailed description.

`verbose` *bool* [Sieve Conf]
: If *bool* is '`true`', log all executed actions.

`line-info` *bool* [Sieve Conf]
: If *bool* is '`true`', print source locations along with action logs. This statement takes effect only if `verbose true` is also set.

`email` *addr* [Sieve Conf]
: Set user e-mail 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.

3.10.1.3 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` outputs 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 *logging*). This option causes `sieve` to output its diagnostics to the given syslog facility.

3.10.1.4 Extending `sieve`

The basic set of sieve actions, tests and comparators may be extended using loadable extensions. The 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 searched 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. `--libdir-prefix` command line option (see Section 3.10.1.1 [Invoking Sieve], page 102), or the `library-path-prefix` configuration statement (see Section 3.10.1.2 [Sieve Configuration], page 104).
   4. Additional search directories specified with the `library-path` statement (see Section 3.10.1.2 [Sieve Configuration], page 104) in Sieve configuration file.
   5. Additional search directories specified with the. `--libdir` command line option (see Section 3.10.1.1 [Invoking Sieve], page 102).
   6. Additional search directories specified with the `#searchpath` Sieve directive (see Section 5.3.2 [#searchpath], page 175).
   7. 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
```

3.11 guimb — A Mailbox Scanning and Processing Language

`Guimb` is an experimental tool that iterates over messages in a mailbox (or several mailboxes), applying a Scheme function to each of them.

A user-defined *scheme module* that supplies the function to apply is specified using the `--source` or `--file` option. The module must define at least the following function:

`guimb-message` *msg* [User function]
: Processes message *msg*. This function can alter the message using Guile primitives supplied by mailutils.

The following function definitions are optional:

`guimb-getopt` *args* [User function]
: If defined, this function is called after `guimb` has finished processing the command line. *args* is a list of unconsumed command line arguments.

  The function is intended to provide a way of configuring the module from the command line.

`guimb-end` [User function]
: If defined, this function is called after all mailboxes have been processed.

In the following example we define a module that prints information about each message is the input mailbox, in a way similar to `frm` utility:

```
(define-module (frm)
  :export (guimb-message))

(use-modules (mailutils mailutils))

(define (guimb-message msg)
  (display (mu-message-get-sender msg))
  (display " ")
  (display (mu-message-get-header msg "subject"))
  (newline))
```

The modules are looked up in directories listed in the global variable `%load-path`. New directories can be added to that variable on the fly using the `-L` (`--load-path`) option. For example, if the sample module above was saved in a file `frm.scm` somewhere in the load path, it can be applied to the current user inbox by running the following command:

```
guimb --file frm
```

Specifying Scheme Program to Execute

The Scheme module that defines message processing functions is given via the following options:

`-s` *module*
`--source` *module*
: Load Scheme code from *module*.

This option stops further argument processing, and passes all remaining arguments as the value of *args* argument to the `guimb-getopt` function, if it is defined.

`-f` *module*
`--file` *module*
: Load Scheme source code from *module*. The remaining arguments are processed in the usual way. When using this option, you can pass additional options and or arguments to the module by enclosing them in `-{` and `-}` options (see [Passing Options to Scheme], page 109).

An experimental option is provided, that evaluates a supplied Scheme expression right after loading the module:

`-e` *expr*
`--expression` *expr*
: Evaluate scheme expression.

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 makes it possible 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`) option, the rest of the command line following the option's argument is passed as the *args* argument to the `guimb-getopt` function, if such function is defined. 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 the `--file` option, the additional arguments can 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 `guimb-getopt` function will get the following argument

```
( '-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.

`-L` *dir*

`--load-path` *dir*
Insert *dir* at the beginning of the `%load-path` list. The argument is either a single directory name, or a list of such names, delimited by '`:`' characters.

`-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.

3.12 mda

GNU local mail delivery agent reads a message from its standard input and delivers it to one or more local recipients listed in the command line. When we speak about *local* recipients, we mean that these are system users that are known to the system that runs `mda`. However, the mailboxes of these users can be local as well as remote ones. `mda` is able to deliver mail to any mailbox format, supported by GNU Mailutils. These formats, among others, include '`smtp://`', '`prog://`' and '`sendmail://`' which are equivalent to forwarding a message over SMTP to a remote node.

`Mda` is also able to process incoming messages using Sieve, Scheme or Python scripts and, based on results of this processing, to take a decision on whether to actually deliver and where to deliver them. Due to its extensive scripting facilities, `mda` offers much more flexibility than other popular MDAs.

3.12.1 Using `mda` with Sendmail.

When used with Sendmail, `mda` must be invoked from the local mailer definition in the `sendmail.cf` file. The flags '`lswS`' must be set for the mailer. These mean: the mailer is local, 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, the '`fn`' flags may be specified to allow `mda` to generate the usual '`From `' envelope instead of the one supplied by `sendmail`.

If you wish to use `mda` with non-local authentication, such as SQL or LDAP, you also need 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/sbin/mda,
        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/sbin/mda')
define(`LOCAL_MAILER_ARGS', `mail $u')
```

3.12.2 Using `mda` with Exim.

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

```
# transport
mda_pipe:
  driver = pipe
  command = /usr/local/sbin/mda $local_part
  return_path_add
  delivery_date_add
  envelope_to_add

# director
```

```
mda:
  driver = localuser
  transport = mda_pipe
```

3.12.3 Using `mda` with MeTA1.

MeTA1 (`http://meta1.org`) communicates with the delivery agent using LMTP. Instead of using `mda` you will have to start the LMTP daemon `lmtpd` and configure MeTA1 to communicate with it. See Section 3.13.1 [MeTA1-lmtpd], page 121, for details.

3.12.4 Mailbox Quotas

Mailbox quota is a limit on the size of the mailbox. When a mailbox size reaches this limit, `mda` stops accepting messages for this recipient and returns an error condition to the sender. The error code is accompanied by the following error message:

user`: mailbox quota exceeded for this recipient`

Furthermore, if accepting the incoming message would make the mailbox size exceed the quota, such a message will be rejected as well. In this case, the error message is:

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

In both cases, the default return code will be '`service unavailable`' (corresponding to the SMTP return code '`550`'), unless the following statement is present in the `maidag` configuration file:

```
quota {
  exit-tempfail yes;
}
```

in which case a temporary error will be returned.

The mailbox quota can be retrieved from the following sources:

1. Authentication method.
2. DBM file.
3. SQL database.

3.12.4.1 Keeping Quotas in DBM File

To use DBM quota database, GNU Mailutils must be compiled with one of the following command line options: `--with-gdbm`, `--with-berkeley-db`, `--with-ndbm`, `--with-tokyocabinet`, or `--with-kyotocabinet`. Examine the output of `mda --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 explicitly listed in the database.

Value — 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 limit for this user.

Here is an example of a quota database in text form:

```
# 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 quota database, specify its absolute name using the `database` configuration statement in the `quota` section, e.g.:

```
quota {
  database /etc/mail/quota.db;
}
```

3.12.4.2 Keeping Quotas in SQL Database

User quotas can be kept in an SQL table as well. Currently (as of mailutils version 3.19) it is assumed that this table can be accessed using the credentials set in '`sql`' configuration statement (see [SQL Statement], page 37).

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 user quota the following query can be used:

```
SELECT quota FROM mailbox_quota WHERE user_name='${user}'
```

To define this query use the `sql-query` statement:

```
quota {
  sql-query "SELECT quota "
            "FROM mailbox_quota "
            "WHERE user_name='${user}'";
}
```

There are 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. `Mda` always uses the first tuple from the set returned by mailbox quota query. So, you can add a special entry to the `mailbox_quota` table that would keep the group quota. In the discussion below we assume that the `user_name` column for this entry is 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 ('${user}','00DEFAULT')
ORDER BY user_name DESC
```

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

The following configuration statement instructs `maidag` to use this query for retrieving the user quota:

```
quota {
  sql-query "SELECT quota "
            "FROM mailbox_quota "
            "WHERE user_name IN ('${user}','00DEFAULT') "
            "ORDER BY user_name DESC";
}
```

3.12.5 Scripting in `mda`

`Mda` can use global or per-user *mail filters* to decide whether to deliver the message, and where to deliver it. As of Mailutils version 3.19, such mail filters may be written in the following languages:

- Sieve See Chapter 5 [Sieve Language], page 171.
- Scheme
- Python

Mail filters to use are specified using '`script`' configuration statement. The following meta-symbols can be used in its argument:

~

%h Expands to the recipient home directory.

%u Expands to the recipient user name.

By default, the filename extension decides which scripting language will be used. User can alter the choice using '`language`' configuration statement. For example:

```
script {
  language python;
  pattern "~/.maidag-py-filter";
}
```

3.12.5.1 Sieve MDA Filters

The file name of the Sieve filter to use is specified using '`script`' configuration statement. For example, the following configuration statement:

```
script {
  pattern "~/.maidag.sv";
}
```

instructs `maidag` to use file `.maidag.sv` in the recipient home directory as a Sieve filter.

Normal message delivery is attempted if execution of the Sieve code ended with `keep` action (either implicit or explicit).

Other Sieve actions are executed as described in Section 5.7 [Actions], page 184. For example, to deliver message to another mailbox, use the `fileinto` action.

Any modifications to headers or body of the message performed by the Sieve code will be visible in the delivered message.

3.12.5.2 Scheme MDA Filters

The file name of the Scheme mail filter is specified using '`script`' configuration statement. For example, the following configuration statement:

```
script {
  pattern "~/.maidag.scm";
}
```

instructs `mda` to use file `.maidag.scm` in the recipient home directory as a Scheme filter.

3.12.5.3 Python MDA Filters

The file name of the Python mail filter is specified using '`script`' configuration statement. For example, the following configuration statement:

```
script {
  pattern "~/.maidag.py";
}
```

instructs `mda` to use the file `.maidag.py` in the recipient home directory as a Python filter.

A simple example of a mail filter written in Python:

```
from mailutils import *
import maidag
import re

msg = message.Message (maidag.message)
hdr = msg.header

try:
    if 'List-Post' in hdr and 'Received' in hdr \
       and hdr['Received'].find ('fencepost.gnu.org') != -1:

        # check envelope's sender address
        m = re.search (r'([\w\-]+)-bounces\+([\w]+)=.*',
                       msg.envelope.get_sender ())
        if m:
            lbox = m.group (1)
            user = m.group (2)
            # open destination mailbox and append message
            dst = mailbox.MailboxDefault ('~/Mail/%s' % lbox)
            dst.open ('ac')
            dst.append_message (msg)
            dst.close ()
            # set deleted flag so maidag will not deliver msg elsewhere
            msg.attribute.set_deleted ()
```

```
except Exception:
    pass
```

3.12.6 Forwarding

A *forward file* is a special file in the user's home directory that contains the email address of the mailbox where the user wants to forward his mail. Normally, forward files are processed by MTA. However, there are some MTA that lack this feature. One of them is MeTA1.

`Mda` provides a forwarding feature that is useful to compensate the lack of it. This feature is controlled by the `forward` section in the configuration file:

```
forward {
  # Process forward file.
  file name;
  # Configure safety checks for the forward file.
  file-checks (list);
}
```

The name of the forward file is given by the `file` statement in the `forward` section. A common usage is:

```
forward {
  file .forward;
}
```

The forward file is always searched in the recipient home directory.

Before actually using the forward file, a number of safety checks are performed on it. If the file fails to pass one of these checks, no forwarding is performed and the message is delivered as usual. These checks are configured using the `forward.file-checks` statement:

```
forward {
  file .forward;
  file-checks (list);
}
```

Its argument is a list of the following keywords:

groupwritablefile
file_iwgrp The file must not be group writable.

worldwritablefile
file_iwoth The file must not be world writable.

linkedfileinwritabledir
link The file cannot be a symlink in a writable directory.

fileingroupwritabledir
dir_iwgrp The file cannot reside in a group writable directory.

fileinworldwritabledir
dir_iwoth The file cannot reside in a world writable directory.

all All of the above checks.

The default is '`file-checks all`'.

Each of these keywords may be prefixed by 'no' to disable this particular check. For example:

```
forward {
  file-checks (nodir_iwoth, nodir_iwgrp);
  file .forward;
}
```

3.12.7 MDA Configuration File Summary

The behavior of mda is affected by the following configuration statements:

Statement	Reference
debug	See Section 3.2.7 [debug statement], page 18.
mailbox	See Section 3.2.8 [mailbox statement], page 19.
locking	See Section 3.2.10 [locking statement], page 22.
pam	See Section 3.2.16 [pam statement], page 34.
sql	See Section 3.2.19 [sql statement], page 37.
virtdomain	See Section 3.2.17 [virtdomain statement], page 34.
radius	See Section 3.2.18 [radius statement], page 35.
ldap	See Section 3.2.20 [ldap statement], page 40.
auth	See Section 3.2.15 [auth statement], page 32.
mailer	See Section 3.2.11 [mailer statement], page 24.

`stderr` *bool* [MDA Config]

If *bool* is true, log to standard error instead of syslog.

`deliver { ... }` [MDA Config]

This section contains general delivery settings:

```
deliver {
  domain string;
  exit-multiple-delivery-success arg;
};
```

`domain` *name* [deliver]

Default email domain.

`exit-multiple-delivery-success` *arg;* [deliver]

In case of multiple delivery, exit with code 0 if at least one delivery succeeded.

`forward { ... }` [MDA Config]

Controls the forwarding support:

```
forward {
  file name;
  file-checks (list);
}
```

`file` *name* [forward]

Defines the name of the forward file. E.g.:

```
forward {
  file .forward;
```

```
}
```

See Section 3.12.6 [Forwarding], page 116, for a detailed description.

`file-checks (`*`list`*`)` [forward]
Configures safety checks to be performed on the forward file prior to using it. See Section 3.12.6 [Forwarding], page 116, for a detailed description.

`quota { ... }` [MDA Config]
This section configures mail quota support. See Section 3.12.4 [Mailbox Quotas], page 112, for a detailed description.

```
quota {
  database name;
  sql-query query;
  exit-tempfail bool;
}
```

`database` *`name`* [quota]
Sets the name of the quota database in DBM format. See Section 3.12.4.1 [DBM Quotas], page 112.

`sql-query` *`query`* [quota]
If the quotas are kept in a SQL table, this statement defines the SQL query to retrieve the quota for a given user name. See Section 3.12.4.2 [SQL Quotas], page 113.

`exit-tempfail` *`bool`* [quota]
By default, if a message cannot be delivered because the user has exceeded its mail quota, or its delivery would cause it to be exceeded, `MDA` exits with the '`service unavailable`' status, which causes MTA to return the 550 code. If `exit-tempfail` is set to true, it will return a temporary error instead.

`script { ... }` [MDA Config]
Controls scripting. See Section 3.12.5 [MDA Scripting], page 114.

```
script {
  language lang;
  pattern glob;
}
```

`language` *`lang`* [script]
Defines the language that is used for scripting. Allowed values for *lang* are: '`sieve`', '`scheme`', or '`python`'. See [scripting language], page 114.

`pattern` *`pat`* [script]
Defines the pattern for the script file name. The '`~`' at the begiining of the pattern will be replaced with the name of the home directory of the recipient user. The '`%u`' in pattern will be replaced with the recipient user name, and '`%h`' with the home directory for that user.

3.12.8 Mailing list implementation

This subsection explains how to implement mailing lists in `mda` using the '`prog`' mailbox scheme.

Delivery to the '`prog`' mailbox results in invoking the specified command with the given arguments and passing the message to its standard input. There are two ways to specify a '`prog`' mailbox:

prog://*program*?*args*
: Here, *program* is the absolute pathname of the program binary, and *args* are its arguments, separated by '`&`' signs.

`|`*program args*
: In this notation, *args* are command line arguments separated by white space.

In both cases, *args* do not include `argv[0]`.

The '`prog`' mailbox can be used to implement mailing lists.

For example, suppose that the `mda` configuration contains:

```
auth {
  authorization (sql, system);
  authentication (generic, system);
}

sql {
  interface mysql;
  db mail;
  getpwnam "SELECT user as name, mailbox, "
           "'x' as passwd, 500 as uid, 2 as gid, "
           "'/nonexistent' as dir, '/sbin/nologin' as shell "
           "FROM userdb "
           "WHERE user='${user}'";
}
```

Then, the following entries in the '`userdb`' table implement the `mailman@yourdomain` mailing list:

```
mysql> select * from userdb;
+---------------------+----------------------------------------+
| user                | mailbox                                |
+---------------------+----------------------------------------+
| mailman             | |/usr/bin/mailman post mailman         |
| mailman-admin       | |/usr/bin/mailman admin mailman        |
| mailman-bounces     | |/usr/bin/mailman bounces mailman      |
| mailman-confirm     | |/usr/bin/mailman confirm mailman      |
| mailman-join        | |/usr/bin/mailman join mailman         |
| mailman-leave       | |/usr/bin/mailman leave mailman        |
| mailman-owner       | |/usr/bin/mailman owner mailman        |
| mailman-request     | |/usr/bin/mailman request mailman      |
| mailman-subscribe   | |/usr/bin/mailman subscribe mailman    |
| mailman-unsubscribe | |/usr/bin/mailman unsubscribe mailman  |
```

3.13 lmtpd

The LMTP is a local mail transport protocol defined in RFC 2033. GNU Mailutils is shipped with `lmtpd` - a daemon for delivering messages using this protocol.

The daemon shares most of its codebase and configuration with `mda` and consequently provides the same features. See Section 3.12 [mda], page 111, for a detailed description of these.

The behavior of `lmtpd` is affected by the following configuration statements:

Statement	**Reference**
server	See Section 3.2.14 [Server Settings], page 28,
acl	See Section 3.2.12 [acl statement], page 25.
tcp-wrappers	See Section 3.2.13 [tcp-wrappers statement], page 27.
debug	See Section 3.2.7 [debug statement], page 18.
mailbox	See Section 3.2.8 [mailbox statement], page 19.
locking	See Section 3.2.10 [locking statement], page 22.
pam	See Section 3.2.16 [pam statement], page 34.
sql	See Section 3.2.19 [sql statement], page 37.
virtdomain	See Section 3.2.17 [virtdomain statement], page 34.
radius	See Section 3.2.18 [radius statement], page 35.
ldap	See Section 3.2.20 [ldap statement], page 40.
auth	See Section 3.2.15 [auth statement], page 32.
mailer	See Section 3.2.11 [mailer statement], page 24.

3.13.1 Using `lmtpd` with MeTA1.

MeTA1 (`http://meta1.org`) communicates with the delivery agent using LMTP.

The socket to listen for LMTP requests must be specified using the `server` statement (see Section 3.2.14 [Server Settings], page 28). For the purposes of this section, let's suppose `lmtpd` will listen on a UNIX socket `/var/spool/meta1/lmtpsock`. Then, the following (minimal) `lmtpd` configuration will do the job:

```
# Run as daemon.
mode daemon;
# Switch to this group after startup.
group meta1c;
# Configure server:
server unix:///var/spool/meta1/lmtpsock {
  transcript no;
};
```

To configure MeTA1 to use this socket, add the following statement to the '`smtpc`' section in `/etc/meta1/meta1.conf`:

```
  LMTP_socket="lmtpsock";
```

3.14 putmail

The `putmail` utility reads a message from its standard input and delivers it to the specified mailbox URL. The usage is:

```
putmail URL
```

For example, to deliver mail to a local mailbox `/var/spool/mail/test`:

```
putmail /var/spool/mail/test
```

Of course, this would work only it the `test` mailbox is writable for the user invoking `putmail`.

The `smtp` mailbox scheme can be used for remote delivery. For example:

```
putmail 'smtp://mail.example.org;to=ovr'
```

The program will initiate SMTP dialog with the server '`mail.example.org`' and will send the message from its standard input to the user '`ovr`' on that server.

3.14.1 putmail options

`-f` *email*
`-r` *email*
`--from=`*email*
: Specify the sender address. If not used, the current user name will be used.

`-l` *name*
`--language=`*name*
: Define scripting language for the next `--script` option. Valid arguments are '`sieve`', '`scheme`' and '`python`'.

`--message-id-header=`*header*
: Use this header to identify messages when logging Sieve actions

`-s` *name*
`--script=`*name*
: Set the name of the user-defined mail filter. See Section 3.12.5 [MDA Scripting], page 114, for a detailed discussion of the scripting feature.

3.14.2 putmail configuration

The behavior of `putmail` is affected by the following configuration statements:

Statement	Reference
debug	See Section 3.2.7 [debug statement], page 18.
mailbox	See Section 3.2.8 [mailbox statement], page 19.
locking	See Section 3.2.10 [locking statement], page 22.
pam	See Section 3.2.16 [pam statement], page 34.
sql	See Section 3.2.19 [sql statement], page 37.
virtdomain	See Section 3.2.17 [virtdomain statement], page 34.
radius	See Section 3.2.18 [radius statement], page 35.
ldap	See Section 3.2.20 [ldap statement], page 40.
auth	See Section 3.2.15 [auth statement], page 32.
mailer	See Section 3.2.11 [mailer statement], page 24.

The utility also accepts all MDA configuration statements: See Section 3.12.7 [Conf-mda], page 117.

3.15 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, Section "mime.types" in *mime.types man page*. 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
```

3.15.1 Mimeview Invocation

The following table summarizes options specific for `mimeview`:

`-a[`*`type-list`*`]`
`--no-ask[=`*`type-list`*`]`
: By default `mimeview` asks for confirmation 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

[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.

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

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

1 Additionally displays each file name along with its MIME type

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

3 Additionally, enables debugging of `mime.types` parser ('`g`').

4 Additionally, enables debugging of `mime.types` lexer ('`l`').

digits from 5 to 9
The same as 4, currently.

If *flags* are not given, the default '`2`' 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.

`-i`
`--identify`
Identifies and prints the MIME type for each input file.

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

`-f` *file*
`--mimetypes` *file*
Use *file* as `mime.types` file. If *file* is a directory, use *file*`/mime.types`

`-t`
`--lint` Check syntax of the `mime.types` file and exit. Command line arguments are ignored.

3.15.2 Mimeview Config

The following configuration statements affect the behavior of `mimeview`:

Statement	**Reference**
debug	See [Debug Statement], page 18.

`mimetypes` *file* [Mimeview Config]

Read *file* instead of the default `mime.types`.

`metamail` *program* [Mimeview Config]

Use *program* to display files.

3.16 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.

The server operation mode is configured using `mode` statement (see Section 3.2.14 [Server Settings], page 28).

3.16.1 Login delay

POP3 clients often login frequently to check for new mail. Each new connection implies authenticating the user and opening his maildrop and can be very resource consuming. To reduce server load, it is possible to impose a minimum delay between any two consecutive logins. This is called '`LOGIN-DELAY`' capability and is described in RFC 2449.

As of version 3.19, GNU Mailutils `pop3d` allows to set global login delay, i.e. such enforcement will affect all POP3 users. If a user attempts to log in before the specified login delay expires, he will get the following error message:

```
-ERR [LOGIN-DELAY] Attempt to log in within the minimum login delay interval
```

The message will be issued after a valid password is entered. This prevents this feature from being used by malicious clients for account harvesting.

To enable the login delay capability, specify the minimum delay using `login-delay` configuration statement, e.g.:

```
login-delay 60;
```

The `pop3d` utility keeps each user's last login time in a special DBM file, called *login statistics database*, so to be able to use this feature, Mailutils must be compiled with DBM support. By default, the login statistics database is called `/var/run/pop3-login.db`. You can change its name using `stat-file` configuration statement:

```
login-delay 60;
stat-file /tmp/pop.login.db;
```

The login delay facility will be enabled only if `pop3d` is able to access the statistics database for both reading and writing. If it is not, it will report this using `syslog` and start up without login delay restrictions. A common error message looks like:

```
Unable to open statistics db: Operation not permitted
```

You can check whether your `pop3d` uses login delays by connecting to it and issuing the '`CAPA`' command. If login delays are in use, there response will contain the string '`LOGIN-DELAY` *n*', where *n* is the actual login delay value.

3.16.2 Auto-expire

Automatic expiration of messages allows you to limit the period of time users are permitted to keep their messages on the server. It is enabled by `expire` configuration statement:

`expire` *n*`;` Enable automatic expiration of messages after *n* days.

The current implementation works as follows. When a message is downloaded by `RETR` or `TOP` command, it is marked with '`X-Expire-Timestamp:` *n*' header, where *n* is current value of UNIX timestamp. The exact expiration mechanism depends on you. Mailutils allows you two options:

1. Expired messages are deleted by `pop3d` upon closing the mailbox. You specify this mechanism using `delete-expired` configuration statement:

   `delete-expired` *bool*`;`
   If *bool* is '`true`', delete expired messages after receiving the `QUIT` command.

2. Expired messages remain in the mailbox after closing it. The system administrator is supposed to run a cron job that purges the mailboxes. Such a cron job can be easily implemented using `sieve` from GNU Mailutils and the following script:

   ```
   require "timestamp";
   # Replace "5" with the desired expiration period
   if timestamp :before "X-Expire-Timestamp" "now - 5 days"
     {
       discard;
     }
   ```

   This script will remove expired messages 5 days after the retrieval. Replace '`5`' with the desired expiration period and make sure it equals the argument to `expire` configuration keyword.

The statement `expire 0` means the client is not permitted to leave mail on the server. It always implies `delete-expired true`.

3.16.3 Bulletins

The bulletin feature allows you to send important announcements to all POP3 users without mailing them. It works by creating a *bulletin source mailbox* and sending the announcements to it.

After a user successfully authenticates, `pop3d` checks the last *bulletin number* the user receives. The bulletin number refers to the number of the bulletin message in the bulletin source mailbox. If the latter contains more messages, these are appended to the user mailbox.

The user last bulletin number can be kept in two places. First, it can be stored in file `.popbull` in his home directory. Secondly, if Mailutils is compiled with DBM support, the numbers can be kept in a DBM file, supplied via `bulletin-db` configuration statement. If both the database and the `.popbull` file are present, the data from the database take precedence.

To enable this feature, use the following configuration statements:

`bulletin-source` *mbox*
Set the URL of the bulletin source mailbox.

`bulletin-db` *file*
: Set the name of the database file to keep last bulletin numbers in.

The following example instructs `pop3d` to look for the bulletin messages in MH folder `/var/spool/bull/mbox` and to keep the database of last delivered bulletin numbers in `/var/spool/bull/numbers.db`:

```
bulletin-source mh:/var/spool/bull/mbox;
bulletin-db /var/spool/bull/numbers.db;
```

3.16.4 Pop3d Configuration

The following configuration file statements affect the behavior of `pop3d`.

Statement	**Reference**
debug	See Section 3.2.7 [debug statement], page 18.
tls	See Section 3.2.21 [tls statement], page 42.
tls-file-checks	See Section 3.2.22 [tls-file-checks statement], page 42.
mailbox	See Section 3.2.8 [mailbox statement], page 19.
locking	See Section 3.2.10 [locking statement], page 22.
logging	See Section 3.2.5 [logging statement], page 17.
pam	See Section 3.2.16 [pam statement], page 34.
sql	See Section 3.2.19 [sql statement], page 37.
virtdomain	See Section 3.2.17 [virtdomain statement], page 34.
radius	See Section 3.2.18 [radius statement], page 35.
ldap	See Section 3.2.20 [ldap statement], page 40.
auth	See Section 3.2.15 [auth statement], page 32.
server	See Section 3.2.14 [Server Settings], page 28.
acl	See Section 3.2.12 [acl statement], page 25.
tcp-wrappers	See Section 3.2.13 [tcp-wrappers statement], page 27.

`tls-mode` *mode* [Pop3d Conf]
: Configure the use of TLS encryption for inetd mode.

  In daemon mode, this statement sets the type of TLS encryption to use in all server blocks that lack the `tls-mode` statement (see Section 3.2.14.2 [Server Statement], page 30).

  Allowed values for *mode* are:

  no
  : TLS is not used. The `STLS` command won't be available even if the TLS configuration is otherwise complete.

  ondemand
  : TLS is initiated when the user issues the appropriate command. This is the default when TLS is configured.

  required
  : Same as above, but the use of TLS is mandatory. The authentication state is entered only after TLS negotiation has succeeded.

  connection
  : TLS is always forced when the connection is established (POP3S protocol).

`undelete` *bool* [Pop3d Conf]
: On startup, clear deletion marks from all the messages.

`expire` *n* [Pop3d Conf]
: Automatically expire read messages after *n* days. See Section 3.16.2 [Auto-expire], page 128, for a detailed description.

`delete-expired` *bool* [Pop3d Conf]
: Delete expired messages upon closing the mailbox. See Section 3.16.2 [Auto-expire], page 128, for a detailed description.

`login-delay` *duration* [Pop3d Conf]
: Set the minimal allowed delay between two successive logins. See Section 3.16.1 [Login delay], page 127, for more information.

`stat-file` *file* [Pop3d Conf]
: Set the name of login statistics file for the `login-delay` facility. See Section 3.16.1 [Login delay], page 127, for more information.

`bulletin-source` *file* [Pop3d Conf]
: Get bulletins from the specified mailbox. See Section 3.16.3 [Bulletins], page 128, for a detailed description.

`bulletin-db` *file* [Pop3d Conf]
: Set bulletin database file name. See Section 3.16.3 [Bulletins], page 128, for a detailed description.

3.16.5 Command line options

The following table summarizes all `pop3d` command line options.

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

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

`--foreground`
: Remain in foreground.

The Mailutils common options are also understood. See Section 3.1.2 [Common Options], page 8.

3.17 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.

3.17.1 Namespace

GNU `imap4d` supports a notion of *namespaces* defined in RFC 2342. A namespace can be regarded as a list of entities, defining locations to which the user has certain access rights. Each entity includes the *prefix*, under which the mailboxes can be found, *hierarchy delimiter*, a character used to delimit parts of a path to a mailbox, and a *directory* on the file system on the server, which actually holds the mailboxes. Among these three values, only first two are visible to the client using the IMAP '`NAMESPACE`' command.

There are three namespaces:

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.

  By default, this namespace contains a single prefix:

```
prefix: ""
delimiter: /
directory: home directory of the user
```

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.

  By default, this namespace is empty.

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, this namespace is empty.

The default values ensure that each user is able to see or otherwise access mailboxes residing in the directories other than his own home.

These defaults can be changed using the `namespace` block statement:

```
namespace name {
    mailbox-mode mode;
    prefix pfx {
      directory path;
      delimiter chr;
      mailbox-type type;
    }
}
```

The *name* argument to the `namespace` statement declares which namespace is being configured. Allowed values are: '`personal`', '`other`', and '`shared`'.

The `mailbox-mode` statement configures the file mode for the mailboxes created within that namespace (provided that the directory permissions allow the user to create mailboxes). The *mode* argument is a comma-delimited list of symbolic mode settings, similar to that used by `chmod`. Each setting begins with a letter '`g`', which means set mode bits for file group, or '`o`', which means set mode bits for other users (note, that there is no '`u`' specifier, since user ownership of his mailbox cannot be changed). This letter is followed by an '`=`' (or '`+`'), and a list of modes to be set. This list can contain only two letters: '`r`' to set read permission, and '`w`' to set write permission.

For example, the following statement sets read and write permissions for the group:

```
mailbox-mode g=rw;
```

The `prefix` statement configures available prefixes and determines their mappings to the server's file system. The *pfx* argument defines the prefix which will be visible to the IMAP client.

The `directory` statement defines the directory in the file system to which *pfx* is mapped. Exactly one `directory` statement must be present in each `prefix` block. The inerpretation of its argument depends on the namespace in which it occurs.

When used in the '`namespace shared`' block, the argument to this statement is interpreted verbatim, as an absolute pathname.

When used in '`namespace personal`' the argument to `directory` statement can contain references to the following variables (see Section 3.2.2 [Variables], page 15):

user Login name of the user.

home Home directory of the user.

For example, the following statement maps the default personal namespace to the directory '`imap`' in the user's home directory:

```
namespace personal {
  prefix "";
  directory "$home/imap";
}
```

If the '`directory`' statement is used within the '`namespace other`' block, its value can contain the '`$user`' and '`$home`' variables as well, but their meaning is different. For the '`other`' namespace, the '`$user`' variable is expanded to the part of the actual reference contained between the prefix and first hierarchy delimiter (or the end of the reference, if no delimiter occurs to the right of the prefix). Correspondingly, '`$home`' expands to the home directory of that user. Consider, for example, the following statement:

```
namespace other {
  prefix "~";
  directory "/var/imap/$user";
}
```

If the client issues the following statement:

```
1 LIST "~smith" "%"
```

then '`$user`' will expand to the string '`smith`' and the server will look for all mailboxes in the directory `/var/imap/smith`.

The `delimiter` statement defines the folder hierarchy delimiter for that prefix. It is optional, the default value being '`"/"`'.

The `mailbox-type` statement declares the type of the mailboxes within that prefix. If present, its argument must be a valid mailbox type (e.g. '`mailbox`', '`maildir`', or '`mh`'). The IMAP `LIST` command will display only mailboxes of that type. The `CREATE` command will create mailboxes of that type.

In the absence of the `mailbox-type` statement, the IMAP `LIST` command will display mailboxes of any type supported by Mailutils. The type of newly-created mailboxes is then determined by the `mailbox-type` statement (see [mailbox-type], page 21).

Any number of `prefix` blocks can be present.

Consider, for example, the following configuration:

```
namespace personal {
   prefix "" {
      directory "$home/mailfolder";
   }
   prefix "#MH:" {
      directory "$home/Mail";
      delimiter "/";
      mailbox-type "mh";
   }
}
```

It defines the personal namespace containing two prefixes. The empty prefix is mapped to the directory `mailfolder` in the home directory of the currently authenticated user. Any type of mailboxes is supported within that prefix.

The prefix '`#MH:`' is mapped to the directory `Mail` in the home directory of the user, and is limited to contain only mailboxes in MH format.

Note that if the prefixes '`""`' is not defined in the personal namespace, the following default will be automatically created:

```
prefix "" {
  directory "$home";
}
```

3.17.2 Configuration of `imap4d`.

The behavior of `imap4d` is altered by the following configuration statements:

Statement	**Reference**
debug	See Section 3.2.7 [debug statement], page 18.
tls	See Section 3.2.21 [tls statement], page 42.
tls-file-checks	See Section 3.2.22 [tls-file-checks statement], page 42.
mailbox	See Section 3.2.8 [mailbox statement], page 19.
locking	See Section 3.2.10 [locking statement], page 22.
logging	See Section 3.2.5 [logging statement], page 17.

pam	See Section 3.2.16 [pam statement], page 34.
sql	See Section 3.2.19 [sql statement], page 37.
virtdomain	See Section 3.2.17 [virtdomain statement], page 34.
radius	See Section 3.2.18 [radius statement], page 35.
ldap	See Section 3.2.20 [ldap statement], page 40.
auth	See Section 3.2.15 [auth statement], page 32.
server	See Section 3.2.14 [Server Settings], page 28.
acl	See Section 3.2.12 [acl statement], page 25.
tcp-wrappers	See Section 3.2.13 [tcp-wrappers statement], page 27.

`namespace` *`name`* { ... } [Imap4d Conf]

Configures namespace. The argument is one of: '`personal`', '`other`', '`shared`'. The following statements (described below) are allowed within curly braces: `mailbox-mode` and `prefix`.

See Section 3.17.1 [Namespace], page 131.

`mailbox-mode` *`mode`* [Imap4d namespace]

Configures the file mode for the mailboxes created within that namespace. The syntax for *mode* is:

```
g(+|=)[wr]+,o(+|=)[wr]+
```

See Section 3.17.1 [Namespace], page 131.

`prefix` *`pfx`* { ... } [Imap4d namespace]

Configures a prefix and determines its mapping to the server's file system. The *pfx* argument is the prefix which will be visible to the IMAP client. Available substatements are: `directory`, `delimiter`, and `mailbox-type`.

See Section 3.17.1 [Namespace], page 131.

`directory` *`path`* [Imap4d namespace.prefix]

Defines the directory in the file system to which the prefix is mapped.

See Section 3.17.1 [Namespace], page 131.

`delimiter` *`chr`* [Imap4d namespace.prefix]

Defines the folder hierarchy delimiter for the prefix. Argument must be a single character.

See Section 3.17.1 [Namespace], page 131.

`mailbox-type` *`type`* [Imap4d namespace.prefix]

Defines the type of the mailboxes inside that prefix.

See Section 3.17.1 [Namespace], page 131.

`login-disabled` *`bool`* [Imap4d Conf]

Disable `LOGIN` command, if *bool* is '`true`'.

`create-home-dir` *`bool`* [Imap4d Conf]

Create nonexistent user home directories. See also home-dir-mode, below.

`home-dir-mode` *mode* [Imap4d Conf]
: Set file mode for created user home directories. Mode is specified in octal.

  The default value for *mode* is '`700`' ('`drwx------`' in `ls` terms).

`preauth` *mode* [Imap4d Conf]
: Configure PREAUTH mode. Valid arguments are:

  prog:///*program-name*
  : `Imap4d` invokes an external program to authenticate the connection. The command line is obtained from the supplied string, by expanding the following meta-variables:

    `${client_address}`
    : Remote IP address in dotted-quad notation;

    `${client_port}`
    : Remote port number;

    `${server_address}`
    : Local IP address;

    `${server_port}`
    : Local port number.

    If the connection is authenticated, the program should print the user name, followed by a newline character, on its standard output and exit with code '`0`'.

    Otherwise, it should exit with a non-zero exit code.

  ident[://:*port*]
  : The remote machine is asked about the requester identity using the identification protocol (RFC 1413). Both plaintext and DES encrypted replies are understood. Optional *port* specifies the port to use, if it differs from the default '`113`'. It can be either a decimal port number or a symbolic name of a service, listed in `/etc/services`.

  stdio
  : PREAUTH mode is enabled automatically if imap4d is started from command line in interactive mode (`-i` command line option). The current login name is used as the user name.

`preauth-only` *bool* [Imap4d Conf]
: If *bool* is '`true`', use only preauth mode. If unable to setup it, disconnect immediately.

`ident-keyfile` *file* [Imap4d Conf]
: Set DES keyfile for decoding encrypted ident responses. Used with '`ident://`' preauth mode.

`ident-encrypt-only` *bool* [Imap4d Conf]
: Use only encrypted IDENT responses.

`id-fields` *list* [Imap4d Conf]
: Set list of fields to return in response to ID command.

Valid field names are:

name Package name (‘`GNU Mailutils`’).

version Package version (‘`3.19`’).

vendor Vendor name (‘`GNU`’).

support-url
The string ‘`http://www.gnu.org/software/mailutils`’

address The string ‘`31 Milk Street, Boston, MA 02196 USA`’.

os OS name.

os-version OS version number.

command Name of the `imap4d` binary.

arguments Invocation command line.

environment
List of environment variables with their values.

3.17.3 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
```

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 20 processes. *Please note*, that there should be no whitespace between the `-d` and its parameter.

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

`--foreground`
Run in foreground.

`--preauth`
Start in preauth mode

`--test` Run in test mode.

See also Section 3.1.2 [Common Options], page 8.

3.18 Comsat Daemon

Comsatd is the server which receives reports of incoming mail and notifies users about it. By default, it prints subject, sender name and email, followed by first five lines of each newly arrived message to the tty of the recipient user. Users can customize this behavior.

3.18.1 Starting `comsatd`

`-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.

`-t[`*file*`]`
`--test[=`*file*`]`
Test mode. In this mode, `comsatd` takes two arguments: URL of a mailbox and QID of the message from that mailbox and prints the notification to the current user tty (`/dev/tty`), or *file*, if it is supplied. If the `~/.biffrc` file exists, it will be used. For example:

```
$ comsatd --test /var/mail/root 34589
```

Notice, that *file* is an optional argument. When supplied, it should follow the short option form immediately, or the long option form after the equals sign, e.g.:

```
$ comsatd --test=logfile /var/mail/root 34589
```

or

```
$ comsatd -tlogfile /var/mail/root 34589
```

`--foreground`
Don't detach from the controlling terminal, remain in foreground.

See also Section 3.1.2 [Common Options], page 8.

3.18.2 Configuring `comsatd`

Following configuration statements affect the behavior of `comsatd`:

Statement	**Reference**
debug	See Section 3.2.7 [debug statement], page 18.
logging	See Section 3.2.5 [logging statement], page 17.
mailbox	See Section 3.2.8 [mailbox statement], page 19.
locking	See Section 3.2.10 [locking statement], page 22.
acl	See Section 3.2.12 [acl statement], page 25.

3.18.2.1 General Settings

These statements control the general behavior of the comsat daemon:

`max-lines` *number* [Comsatd Conf]
: Set maximum number of message body lines to be output.

`allow-biffrc` *bool* [Comsatd Conf]
: Enable or disable processing of user's `.biffrc` file. By default, it is enabled.

3.18.2.2 Security Settings

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

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

`request-control-interval` *duration* [Comsatd Conf]
: Set the request control interval.

`overflow-delay-time` *duration* [Comsatd Conf]
: Set initial amount of time to sleep, after the first overflow occurs.

`overflow-control-interval` *duration* [Comsatd Conf]
: Set overflow control interval. If two consecutive overflows happen within that interval, the overflow-delay-time is doubled.

3.18.3 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 3.18.2 [Configuring comsatd], page 137).

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 [-n] *string* [*string...*]
: Output the arguments to the user's terminal device. If several arguments are given they will be output separated by single spaces. The newline character will be printed at the end of the output, unless the `-n` option is used.

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"
```

The above example can also be written as:

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

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"
```

3.19 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.

3.19.1 Major differences between Mailutils MH and other MH implementations

1. Sequence numbers increase monotonically;

   Message sequence numbers are used as UIDs and thus increase monotonically. This means, in particular, that if your inbox has messages in the range '`X--Y`' and you delete all messages and then incorporate new mail, the first incorporated message will be assigned sequence number '`Y + 1`' (other MH implementations will assign '`1`'). If this behavior bugs you, add the following setting to your `.mh_profile`:

   ```
   Volatile-uidnext: true
   ```

   You can always renumber your messages starting from '`1`' by running

   ```
   folder -pack=1
   ```

2. UUCP addresses are not supported;
3. Mailutils supports a set of new format specifications (see Section 3.19.1.1 [Format String Diffs], page 140);
4. Mailutils provides a set of new profile variables (see Section 3.19.1.2 [Profile Variable Diffs], page 142);
5. All programs recognize `--help` and `--version` options

   These options are recognized only if no other arguments are present in the command line. Abbreviations are not recognized. This makes Mailutils MH implementation compatible with the standard usage for GNU tools.
6. Several programs behave differently (see Section 3.19.1.3 [Program Diffs], page 142);

3.19.1.1 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 142) 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 142) and `isreply` (see [isreply MH function], page 141) 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 142) 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.

3.19.1.2 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 140) 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 141.

3.19.1.3 Differences in MH Program Behavior

`anno`

The prompt in interactive mode is '`Component name:`', instead of '`Enter component name:`' displayed by the RAND `anno`.

If a `-component field` is not specified and standard input is not connected to a terminal, `anno` does not display the prompt before reading the component from the standard input. RAND `anno` displays the prompt anyway.

`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.

`inc`

- The `-moveto` option. The `-moveto` option instructs `inc` to move messages into another folder after incorporating them. This option has effect only if the `-truncate` option has also been specified and the underlying mailbox supports the '`move`' operation. Currently only '`imap`' and '`imaps`' mailboxes support it. For example, the following command moves incorporated messages into the '`archive`' folder:

  ```
  inc -file imaps://imap.gmail.com -moveto=archive
  ```

  The '`moveto`' URL parameter can be used instead of this option, e.g.:

  ```
  inc -file 'imaps://imap.gmail.com;moveto=archive'
  ```

- Multiple sources Mailutils `inc` is able to incorporate messages from several source mailboxes. These are specified via multiple `-file` options, e.g.:

  ```
  inc  -truncate \
       -file 'imaps://imap.gmail.com;moveto=archived' \
       -file pops://mail.gnu.org \
       -file /var/mail/root
  ```

- URL parameters The following additional parameters can be used in the mailbox URLs supplied with the `-file` option:

  `moveto=`*`folder`*
  : Moves incorporated messages into another folder. This was discussed above.

  `nomoveto`
  : Disables the previous `-moveto` option.

  `truncate[=`*`bool`*`]`
  : Controls source mailbox truncation. If *bool* is not given or it is '`yes`', the mailbox will be truncated after successful processing. If *bool* is '`no`', the source mailbox will not be truncated.

`mhl`

The '`ignores`' keyword can be used in variable list. In that case, if its value contains more than one component name it must be enclosed in double-quotes, e.g.:

```
leftadjust,compwidth=9,"ignores=msgid,message-id,received"
```

The above is equivalent to the following traditional notation:

```
leftadjust,compwidth=9
ignores=msgid,message-id,received
```

The '`MessageName`' component is not yet implemented.

Interactive prompting is not yet implemented.

The following format variables are silently ignored: '`center`', '`split`', '`datefield`'.

`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.

`pick`

New command line option `-cflags` allows to control the type of regular expressions used. The option must occur right before `--`*`component pattern`* or equivalent construct (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...
```

`prompter`

1. Prompter attempts to use GNU Readline library, if it is installed. Consequently, arguments to `-erase` and `-kill` option must follow GNU style key sequence notation (see Section "Readline Init File Syntax" in *GNU Readline Library*).

If `prompter` is built without `readline`, it accepts the following character notations:

`\`*nnnn* Here, *n* stands for a single octal digit.

`^`*chr* This notation is translated to the ASCII code '*chr* + 0100'.

2. Component continuation lines are not required to begin with a whitespace. If leading whitespace is not present, `prompter` will add it automatically.

`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.

`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 preceding the non-existent one.
2. The `rmm` utility will unlink messages, if the `rmmproc` profile component has empty value, e.g.:

```
rmmproc:
```

`scan`

The `-file` option is not supported. This option is present in the `nmh` implementation of `scan`.

`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.

3.20 mailutils

The `mailutils` utility is a multi-purpose tool shipped with Mailutils. It can be used for various mail and database-related tasks, as well as an auxiliary tool for compiling and linking programs with Mailutils.

3.20.1 Invocation Syntax

`Mailutils` is a command line tool. Its invocation syntax is:

```
mailutils [options] command [args]
```

where *options* are options that affect the behavior of `mailutils` as a whole, *command* instructs it what it is to do and *args* are any arguments the *command* needs in order to be executed.

The commands are:

2047	Decodes or encodes email message headers.
acl	Tests Mailutils access control lists.
cflags	Shows compiler options needed to compile with Mailutils.
dbm	Invokes a DBM management tool.
;filter	Applies a chain of filters to the input.
help	Displays a terse help summary.
imap	Invokes an IMAP4 client shell (in development).
info	Displays information about Mailutils compile-time configuration.
ldflags	Constructs a `ld`(1) command line for linking a program with Mailutils.
logger	Logs information using Mailutils log facility.
pop	Invokes a POP3 client shell.
query	Queries configuration values.
wicket	Scans wicket for matching URLs

3.20.2 mailutils help

The `mailutils help` command lists all available options and command names along with short descriptions of what each of them does. It is similar to the `mailutils --help` option.

A command name can be supplied as an argument to `help`, in which case it will display a help page for that particular command, e.g.:

```
mailutils help ldflags
```

will output help for the `ldflags` command. It is synonymous to the `--help` option used with that particular command, e.g.: `mailutils ldflags --help`.

3.20.3 mailutils info

The `mailutils info` command displays information about Mailutils compile-time configuration. In normal form its output lists a single configuration flag per line, e.g.:

```
$ mailutils info
VERSION=2.99.93
SYSCONFDIR=/etc
MAILSPOOLDIR=/var/mail/
SCHEME=mbox
LOG_FACILITY=mail
IPV6
USE_LIBPAM
HAVE_LIBLTDL
WITH_GDBM
WITH_GNUTLS
WITH_GSASL
```

A configuration flag can consist either of a single word, indicating that a particular capability has been enabled at compile time, or of a keyword/value pair delimited by an equal sign, which indicates a particular value used by default for that feature. For example, '`IPV6`' means that Mailutils was compiled with support for IPv6, whereas '`SYSCONFDIR=/etc`' means that the default place for configuration files is in `/etc` directory.

Such short output is convenient for using `mailutils info` in scripts to decide whether it is possible to use a given feature. To assist human users, the `--verbose` (`-v`) option is provided. It prints a short description next to each flag:

```
$ mailutils info --verbose
VERSION=2.99.93           - Version of this package
SYSCONFDIR=/etc           - System configuration directory
MAILSPOOLDIR=/var/mail/   - Default mail spool directory
SCHEME=mbox               - Default mailbox type
LOG_FACILITY=mail         - Default syslog facility
IPV6                      - IPv6 support
USE_LIBPAM                - PAM support
HAVE_LIBLTDL              - a portable `dlopen' wrapper library
WITH_GDBM                 - GNU DBM
WITH_GNUTLS               - TLS support using GNU TLS
WITH_GSASL                - SASL support using GNU SASL
```

3.20.4 mailutils cflags

The `mailutils cflags` command shows compiler options needed to compile a C source with Mailutils. It is intended for use in configuration scripts and Makefiles, e.g.:

```
CFLAGS=-g -O2 `mailutils cflags`
```

3.20.5 mailutils ldflags

The `mailutils ldflags` command is a counterpart of `cflags` which is used for linking. It constructs a `ld` command line for linking a program with Mailutils.

When used without arguments, it outputs `ld` arguments which would link only with the core Mailutils library `libmailutils`, e.g.:

```
$ mailutils ldflags
-L/usr/local/lib -lmailutils
```

This command accepts a number of keywords which allow to select a particular subset of Mailutils libraries to link with. In particular, the argument '`all`' instructs it to link in all available libraries:

```
$ mailutils ldflags all
-L/usr/local/lib -lmu_mbox -lmu_mh -lmu_maildir -lmu_imap -lmu_pop \
-lmu_mailer -lmu_compat -lmailutils -lmu_auth -lgsasl -lgnutls -lgcrypt \
-lldap -lgnuradius -lpam -ldl
```

Other available keywords are:

mbox	Link in the UNIX mbox format support.
mh	Link in the MH format support.
maildir	Link in the Maildir format support.
imap	Link in the IMAP protocol support.
pop	Link in the POP protocol support.
mailer	Enable support for mailers.
sieve	Link in the support for Sieve mail filtering language.
dbm	Link in the support for DBM databases (libmu_dbm library).
auth	Link in the Mailutils authentication library.
guile	Provide Guile language bindings.
python	Provide Python language bindings.

3.20.6 mailutils stat

The command `mailutils stat` shows status of a mailbox. The name or URL of the mailbox to operate upon is supplied in the first argument. If not given, the command will display status of the invoking user system mailbox.

```
$ mailutils stat
type: maildir
path: /var/mail/smith
URL: /var/mail/smith
size: 3498
messages: 24
recent messages: 3
first unseen: 20
uidvalidity: 1338543026
next uid: 87
access: 2016-12-15 09:15:08 +0200
```

The output format is controlled by the `--format` (`-c`) option. Its argument is the desired format string, composed of ordinary characters, which are reporduced on standard output verbatim, backslash sequences, and format specifiers, beginning with '`%`'.

Backslash sequences are interpreted as in C.

A *format specifier* consists of a leading '`%`' followed by a letter. Optional ':' may occur between '`%`' and the letter. Its presense instructs the program to print the description of the corresponding value before the value itself.

The following format sequences are understood:

%f Name of the mailbox as supplied in the command line. If `mailutils stat` was used without explicit mailbox argument, '`%f`' is equivalent to '`%U`'.

%t Type of the mailbox ('`mbox`', '`maildir`', etc.). The description string is '`type`'.

%p Path to the mailbox. In case of remote mailboxes, it is the path part of the mailbox URL. Description string: '`path`'.

%U URL of the mailbox. Description string: '`URL`'.

%s Size of the mailbox in octets. Description string: '`size`'.

%c Number of messages in the mailbox. Description string: '`messages`'.

%r Number of recent (unread) messages in the mailbox. Description string: '`recent messages`'.

%u Index of the first unseen message. Description string: '`first unseen`'.

%v The UIDVALIDITY value. Description string: '`uidvalidity`'.

%n The UID value which will be assigned to the new message to be incorporated into the mailbox. Description string: '`next uid`'.

%a Access time of the mailbox, as a number of seconds since the epoch.

%A Access time of the mailbox in human-readable format.

3.20.7 mailutils query

The `mailutils query` command queries values from Mailutils configuration files. It takes one or more configuration paths (see Section 3.2.1.3 [Paths], page 14) as its arguments. On output, it displays the values it found, each value on a separate line. If the requested value is a block statement it is displayed in full. For example, if main configuration file contained:

```
logging {
   syslog yes;
   facility mail;
}
```

Then:

```
$ mailutils query .logging.syslog
syslog yes;
$ mailutils query .logging.syslog .logging.facility
syslog yes;
facility mail;
```

```
$ mailutils query .logging
logging {
  syslog yes;
  facility mail;
};
```

Several command line options allow to modify output format. The `--value` option instructs the command to output only values:

```
$ mailutils query --value .logging.syslog
yes
```

The `--path` option instructs it to print full pathnames for each value:

```
$ mailutils query --path .logging.syslog
logging.syslog: yes
```

The `--program` option instructs `mailutils` to behave as if it was called under another program name. For example, the following command:

```
$ mailutils query --program=pop3d .server.transcript
```

will return the value of the '`.server.transcript`' statement which the `pop3d` utility would see.

By default, `mailutils query` operates on the main configuration file. Another configuration file can be supplied using the `--file` (`-f`) option:

```
$ mailutils query --file /usr/local/etc/file.conf .pidfile
```

3.20.8 mailutils 2047

The `mailutils 2047` command is a filter for decoding or encoding email message headers formatted in accordance with RFC 2047 (see `http://www.faqs.org/rfcs/rfc2047.html`. By default, it operates in encode mode and assumes the '`iso-8859-1`' encoding. If arguments are supplied in the command line, they are treated as the text to operate upon. Otherwise the command acts as a UNIX filter, reading lines from the standard input and printing results on the standard output.

For example:

```
$ mailutils 2047 'Keld Jørn Simonsen <keld@dkuug.dk>'
=?ISO-8859-1?Q?Keld_J=F8rn_Simonsen?= <keld@dkuug.dk>
```

The decode mode can be requested via the `--decode` (`-d`) option:

```
$ mailutils 2047 --decode '=?ISO-8859-1?Q?Keld_J=F8rn_Simonsen?= \
 <keld@dkuug.dk>'
Keld Jørn Simonsen <keld@dkuug.dk>
```

The `--charset` (`-c`) option changes the default character set. It is meaningful both in decode and in encode modes. In decode mode it instructs the utility to convert the output to the given character set. In encode mode it indicates the encoding of the input data, which will be reflected in the resulting string:

```
$ mailutils 2047 --charset=utf-8 'Keld Jørn Simonsen <keld@dkuug.dk>'
=?utf-8?Q?Keld J=C3=B8rn Simonsen <keld@dkuug.dk>?=
```

The `--encoding` (`-E`) option can be used in encode mode to change the output encoding. Valid arguments for this option are: '`quoted-printable`' (the default) or '`base64`'.

The `--newline` (`-n`) option prints an additional newline character after each line of output.

3.20.9 mailutils filter

The `mailutils filter` command applies a chain of filters to the input. The filters to apply and their arguments are given in the command line. The full invocation syntax is:

```
mailutils filter [option] filter-chain
```

The syntax for *filter-chain* in Backus-Naur form follows:

```
<filter-chain> ::= <filter> | <filter-chain> "+" <filter>
<filter> ::= <filter-spec> <ARG>*
<filter-spec> ::= <WORD> | "~" <WORD>
```

where <*WORD*> stands for the filter name and <*ARG*> represents filter arguments. To obtain a list of known filter names, run:

```
mailutils filter --list
```

Filters are applied in the order of their appearance, from left to right and operate in encode mode. The plus sign has the same meaning as pipe in shell. The default mode can be changed using the `--decode` (`-d`) and `--encode` (`-e`) options. Whatever the default mode is, a '`~`' character before filter name reverts the mode for that filter alone.

For example, to encode the contents of file `file.txt` in Base64 run:

```
mailutils filter base64 < file.txt
```

To convert it to base64 and use CRLF as line delimiters, run:

```
mailutils filter base64 + crlf < file.txt
```

The following command will decode the produced output:

```
mailutils filter --decode crlf + base64
```

It can also be written as

```
mailutils filter ~crlf + ~base64
```

The following example converts the input from ISO-8859-2 to UTF-8, quotes eventual '`From`' occurring at the beginning of a line, encodes the result in Base64 and changes line delimiters to CRLF:

```
mailutils filter iconv iso-8859-2 utf-8 + from + base64 + crlf
```

This final example removes UNIX-style comments from the input and joins continuation lines:

```
mailutils filter --decode inline-comment -S '#' + linecon
```

Such invocation can be useful in shell scripts to facilitate configuration file processing.

3.20.10 mailutils acl

The `mailutils acl` command tests Mailutils Access Control Lists. By default it reads ACL from the Mailutils configuration file section '`acl`'. The command takes a list of IP addresses as its arguments, applies the ACL to each of them in turn and prints the result.

To select the ACL to test, two options are provided. The `--file` (`-f`) option supplies the name of configuration file to read instead of the default one. The `--path` (`-p` option supplies the pathname (see Section 3.2.1.3 [Paths], page 14) of the ACL section to use

instead of the default '`.acl`'. For example, to test ACL in section '`server 213.130.1.232`' of file `/etc/pop3d.conf` use:

```
mailutils acl --file=/etc/pop3d.conf \
       --path=/server="213.130.1.232"/acl address
```

As an example of its use, consider file `test.conf` with the following contents:

```
acl {
        deny from 10.10.10.1;
        deny from 10.10.1.0/24;
        log from any "Connect from ${address}";
        allow from 10.0.0.0/8;
        allow from 192.168.1.0/24;
        deny from any;
}
```

Then, running `mailutils acl --file=test.conf 127.0.0.1` you will get:

```
Testing 127.0.0.1:
mailutils: Connect from 127.0.0.1
127.0.0.1: deny
```

More examples:

```
$ mailutils acl --file=test.conf 127.0.0.1 10.10.10.1 \
         10.10.1.3 10.5.3.1 192.168.1.0 192.168.2.0
Testing 127.0.0.1:
mailutils: Connect from 127.0.0.1
127.0.0.1: deny
Testing 10.10.10.1:
10.10.10.1: deny
Testing 10.10.1.3:
10.10.1.3: deny
Testing 10.5.3.1:
mailutils: Connect from 10.5.3.1
10.5.3.1: accept
Testing 192.168.1.0:
mailutils: Connect from 192.168.1.0
192.168.1.0: accept
Testing 192.168.2.0:
mailutils: Connect from 192.168.2.0
192.168.2.0: accept
```

The `mailutils` option `--debug-level` will give you a deeper insight into the address matching algorithm:

```
$ mailutils --debug-level=acl.trace9 acl --file test.conf 127.0.0.1
Testing 127.0.0.1:
mailutils: Checking sockaddr 127.0.0.1
mailutils: 1:deny: Does 10.10.10.1/255.255.255.255 match 127.0.0.1? no;
mailutils: 2:deny: Does 10.10.1.0/255.255.255.0 match 127.0.0.1? no;
mailutils: 3:log: Does any match 127.0.0.1? yes;
mailutils: Expanding "Connect from ${address}";
```

```
mailutils: Expansion: "Connect from 127.0.0.1";.
mailutils: Connect from 127.0.0.1
mailutils: 4:accept: Does 10.0.0.0/255.0.0.0 match 127.0.0.1? no;
mailutils: 5:accept: Does 192.168.0.0/255.255.0.0 match 127.0.0.1? no;
mailutils: 6:deny: Does any match 127.0.0.1? yes;
127.0.0.1: deny
```

See Section 3.3.3 [Debugging Categories], page 45.

3.20.11 mailutils wicket

The `mailutils wicket` command looks up matching URLs in the Mailutils ticket file (by default, `~/.mu-tickets`) and prints them. The URLs to look for are supplied in the command line.

Consider the following ticket file as an example:

```
smtp://foo:bar@*
smtp://bar:baz@gnu.org
*://baz:qux@*
*://quux:bar@gnu.org
```

Now, running `mailutils wicket smtp://bar@gnu.org` will show:

```
smtp://bar@gnu.org: /home/user/.mailutils-tickets:2
```

(where *user* is your login name). This means that this URL matches the line 2 in your `.mailutils-tickets` file. The `wicket` command does not show the actual matching line to avoid revealing eventual security-sensitive information. You can instruct it to do so using the `--verbose` (`-v`) option:

```
$ mailutils wicket -v smtp://bar@gnu.org
smtp://bar@gnu.org: /home/user/.mu-tickets:2: smtp://bar:***@gnu.org
```

As you see, even in that case the tool hides the actual password part by replacing it with three asterisks. If you are working in a secure environment, you can tell `mu wicket` to show passwords as well, by supplying the `-v` option twice.

A counterpart of `--verbose` is the `--quite` (`-q`) option, which instructs `wicket` to suppress any output, excepting error messages. This can be used in scripts, which analyze the `mailutils wicket` exit code to alter the control flow.

The `mailutils wicket` tool exits with code 0 if all URLs were matched and with code 1 if some of them were not matched in the ticket file. If an error occurred, the code 2 is returned.

3.20.12 mailutils dbm

The `mailutils dbm` tool manages DBM files using `libmu_dbm` The invocation syntax is:

```
mailutils dbm subcommand [options] file [keys]
```

or

```
mailutils dbm [options] subcommand file [keys]
```

where *subcommand* selects the operation mode, *options* modify the tool behavior and *file* specifies the DBM file to operate upon. Some *command*s allow for optional *keys* to be specified.

The *file* argument can be either a DBM file name or a Database URL.

3.20.12.1 Create a Database

The `create` subcommand and its synonym `load` instruct the tool to create a new database:

```
mailutils dbm create file.db
```

If the argument file already exists, it will be truncated prior to adding new records to it.

The data to populate the database with are read from the standard input. The `mailutils dbm` command supports several formats for these data, which are discussed later. In the simplest case (a so called '`format 0.0`') each input line must consist of two fields separated by any amount of whitespace. The first field is treated as a key and the second one as the corresponding value.

The usual way to read data from a file is, of course, by redirecting the file to the standard input as in:

```
mailutils dbm create file.db < input.txt
```

There is also a special option for that purpose: `--file` (`-f`). Thus, the following command is equivalent to the one above:

```
mailutils dbm create --file input.txt file.db
```

The `--file` option has the advantage that it allows, in conjunction with another options, for copying input file metadata (owner UID, GID and file mode) to the created database. For example, the following command ensures that the created database file will have the same metadata as the input file:

```
mailutils dbm create --file input.txt --copy-permissions file.db
```

The `--copy-permissions` (`-P`) option is the one that does the job.

There are also other ways to control mode and ownership of the created database, which are described below.

More advanced dump formats (e.g. '`version 1.0`' format) carry additional information about the file, including its original name, ownership and mode. If input is in one of these formats, the file name argument becomes optional. If it is not supplied, the name stored in the input stream will be used. For example, supposing that the file `users.dump` is in format 1.0, the following command suffices to restore the original filename, ownership, mode and, of course, data:

```
mailutils dbm create --file users.dump
```

3.20.12.2 Add Records to a Database

The `add` subcommand adds records to a database. Records are read from the standard input and must be formatted as for `create`:

```
mailutils dbm add file.db
```

If the argument file does not exist, it will be created.

Adding a record with a key which is already present in the database produces an error. To replace existing records, use the `replace` subcommand instead.

The same options that affect the behavior of `create` apply to `add` and '`replace`' as well, e.g.:

```
mailutils dbm replace --file input.txt --copy-permissions file.db
```

3.20.12.3 Delete Records

To delete records, use the `delete` subcommand. It reads a list of keys to delete to be specified as arguments in the command line:

```
mailutils dbm delete file.db foo bar
```

The command above will delete from `file.db` records with keys '`foo`' and '`bar`'.

It is not an error to attempt to delete a key that does not exist in the database, although such use will produce a warning message.

By default, keys are matched literally. It is also possible to use various pattern matching techniques, depending on the option specified.

The `--glob` (`-G`) option instructs the tool to use UNIX globbing pattern matching. For example, the command below will delete all keys starting with '`foo`' and ending with a decimal digit:

```
mailutils dbm delete file.db 'foo*[0-9]'
```

(note the quoting necessary to prevent shell from interpreting the metacharacters itself).

Another option, `--regex` (`-R`) instructs `mailutils` to treat supplied keys as extended regular expressions:

```
mailutils dbm delete --regex file.db 'foo.*[0-9]{1,3}'
```

Both options are affected by the `--ignore-case` (`-i`) option, which turns on case-insensitive matching.

Using pattern matching to delete records can be a risky operation as selecting a wrong pattern will lead to removing wrong records. It is recommended to first use the list mode described below to verify that the patterns match the right keys.

3.20.12.4 List the Database

The `list` command lists the content of the database:

```
mailutils dbm list file.db
```

By default, entire content is listed on the standard output.

If supplied more than one command line argument, this mode treats the rest of arguments after the database file name as the keys to look for and lists only records with these keys:

```
$ mailutils dbm list file.db foo bar
foo 1
bar 56
```

The `--glob` and `--regex` options instruct the tool to use UNIX globbing or extended regular expression matching, correspondingly. These were described in detail above.

3.20.12.5 Dump the Database

The `dump` subcommand dumps the database to the standard output in a format suitable for backup or sending over the network (a version 1.0 format).

```
mailutils dbm dump file.db > file.dump
```

The produced file is suitable for input to the `create` (`load`) command. Among other uses, it provides an easy way to convert databases between various formats supported by

Mailutils. For example this is how to convert the database file `file.db` to the GDBM database `new.db`:

```
mailutils dbm dump file.db | mailutils dbm create gdbm://new.db
```

Both `list` and `dump` subcommands share the same set of options. In fact, they are pretty similar, except that use different defaults. The `list` subcommand is designed to produce a human-readable output, whereas the dump subcommand is oriented towards backup purposes.

3.20.12.6 Dump Formats

As of version 3.19, `mailutils dbm` supports two formats for dumping DBM databases. Both formats are line-oriented. Comments are introduced with a sharp (‘`#`’) sign in the column 0 of a line, followed by at least one white space character (space or tab). Sharp sign followed by a colon (‘`#:`’) introduces a *pragmatic comment*, which carries some additional information to the loader.

The *version 0.0* format is suitable for databases whose records contain only ASCII data. In this format, each record occupies a separate line, which consists of the key and value separated by a single TAB character. Empty lines are ignored. For example:

```
$ mailutils list /etc/mail/users.db
root    guessme
smith   pAssword
qed     fooBar
```

The output in version 0.0 format is human readable and can be used as input to the popauth utility (see *popauth*. However, version 0.0 has serious drawbacks. First of all, it is not suitable for databases that contain binary data. Secondly, it cannot properly handle keys beginning with a sharp sign or containing TAB. The version 1.0 format is free from these drawbacks.

The *version 1.0* dump format begins with a *header* containing important information about the file, such as its file name, ownership and file mode. This information is stored in pragmatic comments and allows `mailutils dbm load` to easily recreate an exact copy of the file. The following comments are defined:

#:version=1.0
: Indicates that the data that follow are in version 1.0 format.

#:filename=*s*
: Original database file name, without directory parts.

#:uid=*n* Owner UID.

#:user=*s* Owner name.

#:gid=*n* Owner GID

#:group=*s*
: Owner group name.

#:mode=*o*
: File mode in octal

Following this header are actual data. Each record is output in two parts: key and value. Each part begins with a '`#:len=n`' construct on a line by itself, where *n* is the length of the data in decimal. This line is followed by one or more lines of the actual data, encoded in base64. The data are formatted so that each line does not exceed 76 bytes in length (not counting the terminating newline). An example of this format follows:

```
# Database dump file created by GNU Mailutils 2.99.93 on
# Tue Nov  1 13:28:03 2011
#:version=1.0
#:file=users.db
#:uid=0,user=root,gid=25,group=mail,mode=640
#:len=6
c21pdGgA
#:len=9
cEFzc3dvcmQA
#:len=5
cm9vdAA=
#:len=8
Z3Vlc3NtZQA=
#:len=4
cWVkAA==
#:len=7
Zm9vQmFyAA==
```

3.20.12.7 Dbm Exit Codes

The table below summarizes exit codes used by `mailutils dbm`:

Code	Symbolic name	Meaning
0	EX_OK	Successful termination
64	EX_USAGE	Command line usage error
65	EX_DATAERR	Error in user-supplied data: the input file is badly formatted, or some of the data supplied in the command line are invalid (e.g. user name, uid or the like), etc.
66	EX_NOINPUT	Cannot open input file
67	EX_NOUSER	No such user or UID when trying to set output file ownership
69	EX_UNAVAILABLE	Operation cannot be performed due to some kind of problem (e.g. access to the file denied, etc.)
70	EX_SOFTWARE	Internal software error
74	EX_IOERR	Input/output error

3.20.13 mailutils logger

The `mailutils logger` tool logs information using Mailutils log facility.

Syntax:

```
mailutils logger [options] [message]
```

The *message* argument, if supplied, gives the text to log. If not supplied, the utility reads lines of text from standard input or a file (if the `--file` option is given) and sends them to log:

```
# Send text to log
$ mailutils logger I am here
# Log each line from file.txt
$ mailutils logger --file file.txt
# Read stdin and log it:
$ mailutils logger
```

The default logging channel is bound to standard error. To bind it to syslog, use the `--syslog` command line option. In that case `mailutils` uses facility '`user`' and priority '`err`'. You can change this by using the `--priority` (`-p`) option. Its argument is either a syslog facility name or facility and severity names separated by a dot. For example, the following invocation will use facility '`auth`', severity '`info`':

```
mailutils logger --priority auth.info
```

The syslog tag can be set using the `--tag` (`-t`) option:

```
mailutils logger --tag myprog
```

The default tag is '`mu-logger`'.

The `--severity` (`-s`) option sets the Mailutils severity level. Its argument can be any of the following: '`debug`', '`info`', '`notice`', '`warning`', '`error`', '`crit`', '`alert`', '`emerg`'.

Finally, the `--locus` (`-l`) option binds log messages to a location in a file. Its argument has the following syntax:

file:*line*[:*col*]

where *file* is the file name, *line* is the line number and optional *col* is the column number in that file.

For example, the following invocation:

```
mailutils logger --locus mailutils.conf:34 Suspicious statement
```

will send the following to the log:

```
mu-logger: mailutils.conf:34: Suspicious statement
```

3.20.14 mailutils pop

The `mailutils pop` command invokes an interactive POP3 client shell. It reads commands from the standard input, executes them and displays the results on the standard output. If the standard input is connected to a terminal, the readline and history facilities are enabled (provided that Mailutils is configured with GNU Readline).

The `mailutils pop` commands form two major groups. POP3 protocol commands interact with the remote POP3 server and display responses obtained from it. These commands are named after their POP3 equivalents. Another group, *internal commands*, are used to configure the shell itself.

POP protocol commands

connect [-tls] *hostname* [*port*]

Open connection to *hostname*. If the `-tls` option is given, TLS encryption (also known as POPS protocol) will be used. If *port* argument is not given, the

command uses port 110 for a plain POP connection or 995 for POPS (if `-tls` is given).

stls
: Start TLS negotiation. This command is valid only after successful unencrypted connection has been initiated (using `connect` without `-tls` argument).

user *name*
: Send user name to the server. The `pass` command must follow.

pass [*password*]
: Send password. This command is valid only after `user`. If the *password* argument is omitted, the shell will ask you to enter it. While entering, both echoing and history recording will be disabled. Use this to avoid compromising your password.

apop *user* [*password*]
: Authenticate with APOP. If the *password* argument is omitted, you will be asked to supply it. While entering, both echoing and history recording will be disabled.

capa [-reread] [*name...*]
: List server capabilities. Any number of arguments is accepted. If given, the shell will display only the named capabilities, otherwise it displays entire list. By default `capa` reuses the response of its previous invocation (if there was any), instead of resending the '`CAPA`' command to the server. To force it do so, use the `-reread` option.

noop
: Send a '`NOOP`' (*no operation*) command to the server.

stat
: Get the mailbox size and number of messages in it.

uidl [*number*]
: Shows unique message identifiers. Without arguments, shows identifiers for each message in the mailbox. If *number* is given, the command returns the UIDL of that particular message only.

list [*number*]
: Lists messages. See above for the meaning of *number*. Each line of the produced listing contains describes a single message and contains at least the message number and size in bytes. Depending on the POP3 server implementation, additional fields may be present. For example, Mailutils `pop3d` can also output number of lines in the message in the additional third field.

retr *number*
: Retrieve a message.

top *msgno* [*number*]
: Display message headers and first *number* (default 5) of lines of its body.

dele *number*
: Mark message for deletion.

rset
: Remove deletion marks.

quit
: Quit pop3 session.

disconnect
: Close existing connection.

Internal commands

verbose [on|off|mask|unmask] [secure [payload]]

Control output verbosity. Without arguments the `verbose` command shows current settings.

The argument '`off`' (the default) turns off all additional output. The '`verbose on`' command enables POP3 protocol tracing output. Additional arguments can be used to provide more verbosity. The '`secure`' argument enables display of user passwords in the trace output and the '`payload`' argument enables showing payload data (e.g. response body sent in the reply to '`RETR`' command, etc.) Thus, the full diagnostics output is obtained by

```
verbose on secure payload
```

The '`mask`' and '`unmask`' arguments allow to disable and enable such additional verbosity. For example, supposing the command above is in action, the following command will suppress the display of user passwords in the traces:

```
verbose mask secure
```

Similarly, `verbose unmask secure` will turn it back again.

prompt *string*

Set command prompt. The argument can contain *variable references* in any of the following forms:

```
$name
${name}
```

where *name* is the variable name. Such references are expanded to the actual value of the variable at the time of expansion. The following variables are defined:

Variable	**Expansion**
user	Login name of the authenticated POP3 user. If the session is not authenticated yet, expands to '`[nouser]`'.
host	Name of the remote host, or '`[nohost]`' if no connection is established.
program-name	Name of the program, as typed on the command line to invoke it.
canonical-program-name	'`mailutils`'
package	'`Mailutils`'
version	Mailutils version number (3.19)
status	Session status. One of: '`disconnected`', '`connected`' or '`logged in`'.

For example:

```
prompt "[${user}@$host "
```

Notice the use of quotes to include the space character in the prompt.

exit

Exit the program.

help [*command*]
? [*command*]
: Without arguments displays a list of commands with possible arguments and short descriptions.

  With one argument, displays a terse description for the given *command*.

history
: Shows command history.

3.20.15 mailutils imap

The `mailutils imap` command invokes an interactive IMAP4 client shell. It reads commands from the standard input, executes them and displays the results on the standard output. The shell is similar to the `mailutils pop` (see Section 3.20.14 [mailutils pop], page 158) shell.

IMAP protocol commands

Most commands in this group correspond (with minor differences) to IMAP commands described in RFC 3501[1].

`connect` [*-tls*] *host* [*port*] [imap command]
: Opens connection to the server *host*. If the `-tls` option is given, TLS encryption (also known as IMAPS protocol) will be used. If *port* argument is not supplied, the command uses port 143 for a plain IMAP connection or 993 for IMAPS (if `-tls` is given).

`capability` [*-reread*] [*name...*] [imap command]
: Lists server capabilities. Any number of *name*s is accepted. If at least one is given, the shell will display only the named capabilities, otherwise it displays the entire list. By default, `capability` reuses the response of its previous invocation (if there was any), instead of resending the CAPABILITY command to the server. To force it do so, use the `-reread` option.

`starttls` [imap command]
: Starts TLS negotiation. This command is valid only after unencrypted connection has been successfully initiated using connect without the `-tls` option.

`login` *user* [*password*] [imap command]
: Logs in to the server as *user* with optional *password*. If the pass argument is omitted, the shell will ask you to enter it. While entering, both echoing and history recording will be disabled. Use this to avoid compromising your password.

`logout` [imap command]
`quit` [imap command]
: Quits the imap session.

`id` [*-test kw*] [*arg...*] [imap command]
: Sends IMAP ID command. See RFC 2971[2], for a discussion of arguments. By default, this command outputs entire ID list. If, however, the `-test` option is given, it will check whether the keyword *kw* is defined and display its value if so.

[1] See `http://www.faqs.org/rfcs/rfc3501.html`.

[2] `http://www.faqs.org/rfcs/rfc2971.html`

`check` [imap command]
: Requests a server checkpoint.

`select` [*mbox*] [imap command]
: Selects the named mailbox. Without argument, selects '`INBOX`'.

`examine` [*mbox*] [imap command]
: Examines the named mailbox, i.e. selects it in read-only mode. If *mbox* is not given, '`INBOX`' is assumed.

`status` *mbox kw* [*kw...*] [imap command]
: Gets mailbox status. Valid keywords (*kw* arguments) are: '`MESSAGES`', '`RECENT`', '`UIDNEXT`', '`UIDVALIDITY`', and '`UNSEEN`'. Keywords are case-insensitive.

`fetch` *msgset items* [imap command]
: Fetches message data. See RFC 3501, section 6.4.5[3], for a discussion of its arguments.

`store` *msgset items* [imap command]
: Alters mailbox data. See RFC 3501, section 6.4.6[4], for a discussion of its arguments.

`close` [imap command]
: Closes the currently selected mailbox (with expunge).

`unselect` [imap command]
: Closes the currently selected mailbox (without expunge).

`delete` *mbox* [imap command]
: Deletes the mailbox *mbox*.

`rename` *old-name new-name* [imap command]
: Renames existing mailbox *old-name* to *new-name*.

`expunge` [imap command]
: Permanently removes messages marked for deletion.

`create` *name* [imap command]
: Creates new mailbox with the given *name*.

`append` [*-time datetime*] [*-flag flag*] *mailbox file* [imap command]
: Reads an RFC-822 message from *file* and appends it to the *mailbox*. Use the `-time` option to supply envelope date for the message. Use the `-flag` option to supply message flags. For example:

```
append -time "25-Aug-2002 18:00:00 +0200" -flag \Seen INBOX input.msg
```

`list` *ref mbox* [imap command]
: Lists matching mailboxes. See RFC 3501, section 6.3.8[5], for a discussion of its arguments.

`lsub` *ref mbox* [imap command]
: Lists subscribed mailboxes (RFC 3501, section 6.3.9[6]).

[3] `http://tools.ietf.org/html/rfc3501#section-6.4.5`
[4] `http://tools.ietf.org/html/rfc3501#section-6.4.6`
[5] `http://tools.ietf.org/html/rfc3501#section-6.3.8`
[6] `http://tools.ietf.org/html/rfc3501#section-6.3.9`

`subscribe` *mbox* [imap command]
: Subscribes to a mailbox.

`unsubscribe` *mbox* [imap command]
: Removes mailbox *mbox* from the subscription list.

`noop` [imap command]
: Sends a *no operation* command.

`disconnect` [imap command]
: Closes existing connection.

Internal commands

The `imap` shell implements the same set of internal commands as `pop` shell: See Section 3.20.14 [mailutils pop], page 158. There is only one imap-specific internal command:

`uid` [*on*|*off*] [imap command]
: Controls the UID mode. When the UID mode is on, the commands `fetch` and `store` operate on and return message UIDs instead of their sequence numbers.

  To examine the current state of the UID mode, issue the `uid` command without arguments.

3.20.16 mailutils send

Reads an RFC-822 message from a file and sends it over to a specified SMTP server. The syntax is:

```
mailutils send [options] host file
```

where *host* defines the SMTP server through which to send the message, and *file* is the name of the input file containing the message. For example, to send a message from file `input.msg` using SMTP service at localhost, one would write:

```
$ mailutils send localhost input.msg
```

The *host* argument can be an IP address, hostname, or a valid SMTP URL.

The following command line options are understood:

`-F` *address*
`--from=`*address*
: Supplies envelope sender address.

`-T` *address*
`--rcpt=`*address*
: Supplies envelope recipient address. It can be specified multiple times.

`-t`
`--read-recipients`
: Instructs the program to read recipient email addresses from the message '`To:`', '`Cc:`', and '`Bcc:`' headers.

3.20.17 mailutils smtp

The `mailutils smtp` command invokes an interactive SMTP client shell. It reads commands from the standard input, executes them and displays the results on the standard output. If the standard input is connected to a terminal, the readline and history facilities are enabled (provided that Mailutils is configured with GNU Readline).

Initializing connection

`connect` [*-tls*] *host* [*port*] [smtp command]
: Connects to SMTP server at *host* (IP address or host name). If the `-tls` option is given, TLS encryption (also known as SMTPS protocol) will be used. The default port number is 25 for plain SMTP and 465 for SMTPS. Explicit *port* argument overrides the default value.

Connection parameters

A number of parameters is associated with an open connection:

domain
: Domain name used in EHLO statement. Defaults to the current host name.

The following parameters are used for ESMTP authentication:

username
: User name.

password
: User password.

service
: GSASL service name.

realm
: Realm name.

host
: Host name.

url
: SMTP URL. It can contain all of the above. Default is smtp://

These parameters are manipulated using the following statements:

`set` *param value* [*param value*...] [smtp command]
: Sets parameter *param* to *value*. Several parameters can be set with one `set` statement.

`clear` [*param*...] [smtp command]
: Unset the supplied connection parameters. If used without arguments, unsets all parameters.

`list` [*param*...] [smtp command]
: Lists the values of the connection parameters. If used without arguments, lists all parameters.

SMTP commands

`ehlo` [*domain*] [smtp command]
: Sends the ESMTP greeting. Unless *domain* is supplied, the connection parameter '`domain`' is used.

`capa` [*name*...] [smtp command]
: Lists the server capabilities.

`starttls` [smtp command]
: Initiates encrypted connection. This command is disabled if the connection is opened with the `-tls` option.

`auth` *mech* [*mech...*] [smtp command]
: Authenticate using the supplied mechanisms.

`rset` [smtp command]
: Reset the session state.

`from` [*email*] [smtp command]
: Sets sender email address. If used without arguments, prints the sender email address.

`to` [*email*] [smtp command]
: Sets recipient email address. If used without arguments, prints all recepient names collected so far.

`smtp` *command* [*args...*] [smtp command]
: Sends the *command* with its arguments verbatim.

`quit` [smtp command]
: Quits the SMTP session.

`send` [*file*] [smtp command]
: Reads the message from *file* and sends it. If *file* is not supplied, the action depends on whether a `send` command was used prevously within the same session. If so, `mailutils` will first ask whether to reuse the already supplied message. If not, it will start an editor, allowing you to enter the new message. When you exit from the editor, you will be prompted what to do with the message: send, edit, or quit (discard) it.

Internal commands

Internal commands are the same as in `pop` shell: See Section 3.20.14 [mailutils pop], page 158.

3.20.18 mailutils maildir_fixup

This command fixes attributes and UID assignments in '`maildir`' mailboxes created by mailutils versions prior to 3.10.90.

Attribute flags used in '`maildir`' mailboxes by these versions of mailutils were a bit different from those described in the original description of the '`maildir`' format[7] and those used by another implementations. The discrepancy has been reported in the Mailutils bug tracker[8] and was fixed in version 3.10.90. Along with this fix, measures has been taken to ensure persistence of UID assignments between different sessions. Starting from version 3.10.90, whenever `mailutils` library opens a maildir mailbox, it determines the version that created it. If the mailbox is writable and the library determines that the mailbox is

[7] `http://cr.yp.to/proto/maildir.html`

[8] `http://savannah.gnu.org/bugs/?56428`

affected by the two problems described above, it fixes the mailbox on the fly. This process is completely transparent to the user.

If you operate a site with a large number of mailboxes in '`maildir`' formats, you may choose to fix up all of them at once. That's what the `maildir_fixup` command is for. It takes one or more directory names as its arguments and recursively scans these directories in search for '`maildir`' mailboxes. Each mailbox found is analyzed and a fix-up is performed, if necessary. If a mailbox is already in the new format, it remains untouched.

The following options modify the program's behavior:

`-v`
`--verbose`
: List each maildir name before processing it.

`-n`
`--dry-run`
: Don't touch maildirs, just print their names,

The `maildir_fixup` tool reads main mailutils configuration file by default. It looks for program-specific settings in the section '`program maildir_fixup`'. If the `include` statement is present that has a directory name as its argument, the file `maildir_fixup` is looked up in that directory and parsed, if present.

The program uses the following configuration statements:

Statement	**Reference**
debug	See Section 3.2.7 [debug statement], page 18.
locking	See Section 3.2.10 [locking statement], page 22.
mandatory-locking	See *mandatory-locking statement*.

3.21 dotlock

A stand-alone mailbox-locking utility. It is the default program used by `mailutils` if the `locking.type` configuration statement is set to `external` (see [external locking type], page 23).

The program usage syntax is:

```
# To lock mbox:
dotlock options mbox
# To unlock it:
dotlock -u options mbox
```

By default the program implements the '`dotlock`' locking (see [dotlock locking type], page 23). This can be changed either in the configuration file, or via the command line options.

The following common configuration statements affect the behavior of `dotlock`:

Statement	**Reference**
debug	See [Debug Statement], page 18.
locking	See [Locking Statement], page 22.

The program understands the following command line options:

`-d`
`--debug` Print details of failure reasons to stderr.

`-f[n]`
`--force[=n]`
If a lock file exists and is more than *n* minutes old, forcibly remove it and re-lock the mailbox. Default *n* is 10 minutes.

`-p`
`--pid-check`
Check if the PID of lock owner is still active. If not, break the lock.

`-r n`

`--retry=n`
Number of times to retry acquiring the lock, if it is held by another process. The default is 10 times.

`-t n`
`--delay=n`
Sets delay in seconds between two successive locking attempts. The default is 1 second.

`-u`
`--unlock` Unlock the mailbox.

4 Mailutils Libraries

==========

Editor's note:

This node is to be written.

==========

5 Sieve Language

The input language understood by the GNU Sieve Library is a superset of the Sieve language as described in RFC 3028.

5.1 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, that begins 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 automatic 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 5.3 [Preprocessor], page 175), 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 libmu_sieve 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"];
  ```

5.2 Syntax

Being designed for the sole purpose of filtering mail, Sieve has a very simple syntax.

5.2.1 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 5.1 [Lexical Structure], page 171) 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.

5.2.2 Actions Described

An *action* is a Sieve command that performs some operation over a 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 5.7 [Actions], page 184, for detailed discussion of actions.

5.2.3 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.

5.2.4 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 5.6 [Tests], page 177.

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;
}
```

5.3 Preprocessor

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`.

5.3.1 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.

5.3.2 Sieve #searchpath directive

The `#searchpath` directive adds its argument to the list of directories searched for loadable modules. It has the same effect as `library-path` Sieve configuration statement (see Section 3.10.1.2 [Sieve Configuration], page 104).

5.4 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 need not be explicitly 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 libmu_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. 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, `libmu_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, `libmu_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. `libmu_sieve` then issues the following diagnostic message:

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

5.5 Comparators

GNU libmu_sieve 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.

5.6 Tests

This section describes the built-in tests supported by GNU libmu_sieve. 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'
```

GNU Sieve supports two kinds of tests. *Built-in tests* are defined within the library and do not require any external files. *External tests* are loadable modules that can be linked in at run time using the `require` statement (see Section 5.4 [Require Statement], page 175).

5.6.1 Built-in Tests

`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`] *limit*(`number`) [Test]

The `size` test deals with the size of a message. The required argument *limit* 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*] [Test]
`envelope-part`(`string-list`) *key-list*(`string-list`)

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*(`string-list`) [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] [Test]
header-names(string-list) *key-list*(string-list)

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
```

5.6.2 External Tests

`numaddr` [`:over` | `:under`] *header-names*(string-list) *count*(number) [Test]

Synopsis:

```
require "test-numaddr";
...
if numaddr args
  {
    ...
  }
```

Description: 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 *count*, the test is true; otherwise, it is false.

If the tagged argument is '`:under`' and the number of addresses is less than *count*, the test is true; otherwise, it is false.

If the tagged argument is not given, '`:over`' is assumed.

`pipe` [`:envelope`] [`:header`] [`:body`] [`:exit` *code*(number)] [`:signal` *code*(number)] *command*(string) [Test]

Synopsis:

```
require "test-pipe";

if pipe command
  {
    ...
  }
```

Description: The `pipe` test executes a shell command specified by its argument and pipes the entire message (including envelope) to its standard input. When given, tags `:envelope`, `:header`, and `:body` control what parts of the message to pipe to the command.

In the absence of the `:exit` tag, the test returns true if the command exits with code 0. If `:exit` is given, the test returns true if the command exits with code equal to its argument.

The `:signal` tag determines the result of the test in case if the program exits on signal. By default, the test returns false. If `:signal` is given and the number of signal which caused the program to terminate matches its argument, the test returns true.

`spamd` [`:host` *tcp-host*(string)] [`:port` *tcp-port*(number)] [`:socket` *unix-socket*(string)] [`:user` *name*(string)] [`:over` | `:under` *limit*(string)] [Test]

Synopsis:

```
require "test-spamd";
...
if spamd args
  {
    # This is spam
    ...
  }
```

Description: This test is an interface to SpamAssassin filter. It connects to the `spamd` daemon using connection parameters specified by tagged arguments `:host` and `:port` (if the daemon is listening on an INET socket), or `:socket` (if the daemon is listening on a UNIX socket) and returns true, if SpamAssassin qualifies the message as spam.

Tagged argument *limit* alters the default behavior. Its value is a string representation of a floating point number. If the tag `:over` is used, then the test returns true if the spam score returned from SpamAssassin is greater than *limit*. Otherwise, if `:under` is used, the test returns true if the spam score is less than *limit*. The comparison takes into account three decimal digits.

Tagged argument `:user` allows to select a specific user profile. If it is not given, the user name is determined using the effective UID.

Before returning, the `spamd` test adds the following headers to the message:

X-Spamd-Status
: '`YES`' or '`NO`', depending on whether the message is qualified as spam or ham.

X-Spamd-Score
: Actual spam score value.

X-Spamd-Threshold
: Spam score threshold, as configured in SpamAssassin settings.

X-Spamd-Keywords
: Comma-separated list of keywords, describing the spam checks that succeeded for this message.

Example:

```
request "test-spamd";

if spamd :host 127.0.0.1 :port 3333
  {
     discard;
  }
```

`list` [*comparator*] [*match-type*] [`:delim` *delimiters*(string)] *headers*(string-list) *keys*(string-list) [Test]

Synopsis:

```
require "test-list";
if list args
  {
     ...
  }
```

Description: The `list` test evaluates to true if any of *headers* matches any key from *keys*. Each header is regarded as containing a list of keywords. By default, comma is assumed as list separator. This can be overridden by specifying the `:delim` tag, whose value is a string consisting of valid list delimiter characters.

Example:

This test can be used in conjunction with the `spamd` test described above:

```
require ["fileinto", "test-spamd", "test-list"];

if spamd :host 127.0.0.1 :port 3333
  {
```

```
    if list :matches :delim " ,"
            "X-Spamd-Keywords" [ "HTML_*", "FORGED_*" ]
      {
         fileinto "~/mail/spam";
      }
    else
      {
         discard;
      }
  }
```

`timestamp` [`:before` | `:after`] *header*(`string`) *date*(`string`) [Test]

Synopsis:

```
require "test-timestamp";

if timestamp arg
  {
     ...
  }
```

Description: The `timestamp` test compares the value of a structured date header field (*header*) with the given date (*date*).

If the tagged argument is `:after` and the date from the header is after the specified date the result is true, otherwise, if the header date is before the given date, the result is false.

If the tagged argument is `:before` and the date from the header is before the specified date the result is true, otherwise, if the header date is after the given date, the result is false.

If no tagged argument is supplied, `:after` is assumed.

Almost any date format is understood. See Appendix B [Date Input Formats], page 205, for a detailed information on date formats.

Example:

The test below succeeds if the date in '`X-Expire-Timestamp`' header is more than 5 days older than the current date:

```
require "test-timestamp";

if timestamp :before "X-Expire-Timestamp" "now - 5 days"
  {
     discard;
  }
```

5.7 Actions

There are two groups of GNU Sieve actions: *built-in actions*, which are defined within the library, and *external actions*, i.e. loadable modules that can be linked in at run time using the `require` statement (see Section 5.4 [Require Statement], page 175).

5.7.1 Built-in Actions

The GNU libmu_sieve supports the following built-in 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` [*:permissions mode*] *folder* [Action]

Required arguments:

folder A string representing the folder name

Tagged arguments:

`:permissions` *mode*
: Specifies the permissions to use, if the mailbox is created.

The `fileinto` action delivers the message into the specified folder. If the folder is local, it is created using permissions '`0600`', for regular files, and '`0700`' for directories. This default can be changed by using the `:permissions` tag. Its argument is a mode specification, similar to that used by `chmod` shell utility. It is a list of permissions settings separated by commas. Each setting begins with one of the following letters:

g Set permissions for the users in the file group.

o Set permissions for users not in the file's group.

This letter must be followed by either '`+`' or '`=`' and the list of permissions to be set. This latter list is a string containing any one or both of the following characters:

r Grant permission to read.

w Grant permission to write.

For example, the following instruction creates the mailbox `~/shared` which will be world readable and writable for the group:

```
fileinto :permissions "g=rw,o=r" "~/shared"
```

Notice that:

1. The `:permissions` setting are affected by the current umask value.
2. Only `r` and `w` permissions can be set, since other permissions do not seem to be useful for mailboxes. However, for mailboxes that have a directory structure (such as maildir and MH), any settings in '`g`' and '`o`' sets imply setting the executable bit.
3. Owner's permissions cannot be set. The owner always has all permissions on the mailbox he created.
4. The `:permissions` settings apply only to local mailboxes. They are ignored for remote mailboxes.

`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.

5.7.2 External Actions

GNU Mailutils is shipped with a set of external Sieve actions. These actions are compiled as loadable modules and must be required prior to use (see Section 5.4 [Require Statement], page 175).

`moderator [:keep] [:address` *address*`(string)] [:source` *sieve-file*`(string)] [:program` *sieve-text*`(string)]` [Action]

Synopsis:

```
require "moderator"
moderator args;
```

Description: This action is a moderator robot for Mailman-driven mail archives. A Mailman moderation request is a MIME message consisting of the following three parts:

N	Content-Type	Description
1	text/plain	Introduction for the human reader.
2	message/rfc822	Original submission.
3	message/rfc822	Mailman control message.

Replying to part 3 (keeping the subject intact) instructs Mailman to discard the original submission.

Replying to part 3 while adding an 'Approved:' header with the list password in it approves the submission.

The `moderator` action spawns an inferior Sieve machine and filters the original submission (part 2) through it. If the inferior machine marks the message as deleted, the action replies to the control message, thereby causing the submission to be discarded. The '`From:`' address of the reply can be modified using `:address` tag. After discarding the message, `moderator` marks it as deleted, unless it is given `:keep` tag.

If the `:source` tag is given, its argument specifies a Sieve source file to be used on the message. Otherwise, if `:program` is given, its argument supplies a Sieve program to be used on this message. At most one of these tags may be specified. Supplying them both, or supplying several instances of the same tag, is an error. The behavior of the action in this case is undefined.

If neither `:program` nor `:source` is given, `moderator` will create a copy of the existing Sieve machine and use it on the message.

The action checks the message structure: it will bail out if the message does not have exactly 3 MIME parts, or if parts 2 and 3 are not of '`message/rfc822`' type. It is the responsibility of the caller to make sure the message is actually a valid Mailman moderation request (see the example below).

Example:

```
if allof(header :is "Sender" "mailman-bounces@gnu.org",
         header :is "X-List-Administrivia" "yes")
  {
```

```
            moderator :source "~/.sieve/mailman.sv";
        }
```

`pipe` [`:envelope`] [`:header`] [`:body`] *command*(`string`) [Action]

Synopsis:

```
require "pipe";

pipe command
```

Description: The `pipe` action executes a shell command specified by its argument and pipes the entire message (including envelope) to its standard input. When given, tags `:envelope`, `:header`, and `:body` control what parts of the message to pipe to the command.

Example: The example below uses the `putmail` utility (see Section 3.14 [putmail], page 122) to forward the message to user '`gray`' on the machine '`mail.gnu.org`'.

```
require "pipe";

pipe "/usr/bin/putmail smtp://gray@mail.gnu.org"
```

`vacation` [`:days` *ndays*(`number`)] [`:subject` *subject*(`string`)] [Action]
[`:aliases` *addrlist*(`string-list`)] [`:noreply`
noreply-address(`string-list`)] [`:reply_regex` *expr*(`string`)]
[`:reply_prefix` *prefix*(`string`)] [`:sender` *email*(`string`)]
[`:database` *path*(`string`)] [`:return_address` *email*(`string`)]
[`:header` *headers*(`string-list`)] [`:mime`] [`:always_reply`] [`:rfc2822`]
[`:file`] *text*(`string`)

Syntax:

```
require "vacation";
vacation args;
```

Description: The `vacation` action returns a message with *text* to the sender. It is intended to inform the sender that the recipient is not currently reading his mail.

If the `:file` tag is present, *text* is treated as the name of the file to read the body of the reply message from. When used together with tag `:rfc2822`, the file should be formatted as a valid RFC 2822 message, i.e. headers followed by empty line and body. Headers may not contain '`To`', '`From`', and '`Subject`', as these will be generated automatically.

If the `:subject` tag is given, its argument sets the subject of the message. Otherwise, the subject is formed by prefixing original subject with '`Re:`', or the *prefix* given with the `:reply_prefix` tag. Before prefixing, any original prefixes matching extended regular expression *expr* (`:reply_regex` tag) are stripped from the subject line. If `:reply_regex` is not specified, the default regexp is '`^re: *`'.

Another headers can be added using the `:header` tag. Its argument is a list of header strings, each one having the form '"*name*`:`*value*"'. Additional whitespace is allowed on both sides of the colon.

The `:aliases` tag instructs `vacation` to handle messages for any address in *addrlist* in the same manner as those received for the user's principal email.

Before processing, `vacation` compares the sender address with its *address exclusion list*. Elements of this list are extended case-insensitive regular expressions. If the sender address matches any of these expressions, the message will not be replied. The default exclusion list is:

```
.*-REQUEST@.*
.*-RELAY@.*
.*-OWNER@.*
^OWNER-.*
^postmaster@.*
^UUCP@.*
^MAILER@.*
^MAILER-DAEMON@.*
```

New entries can be added to this list using `:noreply` tag.

The `:days` tag sets the *reply interval*. A reply is sent to each sender once in *ndays* days. GNU Sieve keeps track of sender addresses and dates in file `.vacation` stored in the user's home directory. The file name can be changed using the `:database` tag.

The tag `:always_reply` instructs vacation to respond to the message regardless of whether the user email is listed as a recipient for the message.

5.8 Extensions

The following extensions are implemented

5.8.1 The encoded-character extension

The '`encoded-character`' extension complies with *RFC 5228*, part 2.4.2.4. It provides a way of incorporating multibyte sequences in a Sieve script using only ASCII characters. This is a built-in extension. It is enabled using the following statement:

```
require "encoded-character";
```

When this extension is enabled, the sequences '`${hex: ...}`', and '`${unicode: ...}`' can appear inside of quoted strings.

The sequence

```
${hex: XX}
```

where *XX* is a sequence of one or two-digit hex numbers separated by any amount of whitespace, is replaced with the octets with the hexadecimal values given by each hex number. For example,

```
"${hex: 24 24}" ⇒ "$$"
```

Thus, the following script will discard any message containing three contiguous dollar signs in its '`Subject`' header:

```
require "encoded-character";

if header :contains "Subject" "$${hex:24 24}" {
     discard;
}
```

The '`hex:`' keyword is case-insensitive. If *XX* contains invalid hex numbers, the entire sequence is left verbatim. This is illustrated by the following example:

```
"$${hex:40}"          ⇒ "$@"
"${hex: 40 }"         ⇒ "@"
"${HEX: 40}"          ⇒ "@"
"${hex:40"              ⇒ "${hex:40"
"${hex:400}"          ⇒ "${hex:400}"
"${hex:4${hex:30}}"   ⇒ "${hex:40}"
```

The sequence

```
${unicode: HEXNUM}
```

where *HEXNUM* is a list of hexadecimal numbers separated with whitespace, will be replaced by the UTF-8 encoding of the specified Unicode characters, which are identified by the hexadecimal value of *HEXNUM*. For example, the following string represents a single '`@`' sign:

```
"${UNICODE:40}"
```

Similarly to '`hex:`', the '`unicode:`' indicator is case insensitive. The following examples demonstrate the handling of several valid and invalid encodings:

```
"${unicode:40}"       ⇒ "@"
"${ unicode:40}"      ⇒ "${ unicode:40}"
"${UNICODE:40}"       ⇒ "@"
"${UnICoDE:0000040}" ⇒ "@"
"${Unicode:40}"       ⇒ "@"
"${Unicode:Cool}"     ⇒ "${Unicode:Cool}"
"${unicode:200000}"   ⇒ error
"${Unicode:DF01}      ⇒ error
```

5.8.2 The relational extension

The '`relational`' extension complies with *RFC 3431*. It is a built-in extension. When enabled, the two new match types become available: `:count` and `:value`. Both keywords take a single argument defining the relational operator to use:

'`"gt"`'	greater than ('`>`')
'`"ge"`'	greater than or equal ('`>=`')
'`"lt"`'	less than ('`<`')
'`"le"`'	less than or equal ('`<=`')
'`"eq"`'	equal to ('`==`')
'`"ne"`'	not equal to ('`!=`')

The `:value` keyword requires a relational comparison between strings. The left side of the relation is formed by the value from the message. The right side of the relation is the value from the test expression. If there are multiple values on either side or both sides, the test is considered true if any pair is true. For example,

```
require ["relational", "fileinto"];

if header :value "gt" :comparator "i;ascii-numeric"
                 ["x-spam-level] ["5"]
```

```
{
  fileinto "spam";
}
```

The `:count` keyword counts the specified entities in the message and compares their number with the value given in the test expression. The latter must be a list of one element. This match type can only be used with numeric comparators. For example, the following script will discard any message with 10 or more recipient addresses in the '`To`' and '`Cc`' headers:

```
require "relational";

if address :count "ge" :comparator "i;ascii-numeric"
                       ["to", "cc"] ["10"]
{
    discard;
}
```

5.8.3 The variables extension

The '`variables`' extension is defined in *RFC 5229*. It is a built-in extension. It introduces support for variables in Sieve scripts.

There are two kind of variables: user-defined and match variables.

A *user-defined* variable is initialized using the `set` action:

`set` [*modifiers*] *name*(`string`) *value*(`string`) [Action]

Stores the specified *value* in the variable identified by *name*. Optional *modifiers* are applied on *value* before it is stored in the variable.

The following modifiers are available:

`:lower` Convert value to lower case letters.

`:upper` Convert value to upper case letters.

`:lowerfirst`
Convert the first character in value to lower case.

`:upperfirst`
Convert the first character in value to upper case.

`:quotewildcard`
Quote wildcard characters ('`*`', '`?`', '`\`') by prefixing each occurrence with a backslash ('`\`'). This can be used to ensure that the variable will only match a literal occurrence if used as a parameter to `:matches`.

`:length` The value is the decimal number of characters in the expansion, converted to a string.

When several modifiers are present, they are applied in the following order of precedence (largest value first):

precedence	**modifiers**
40	`:lower` or `:upper`
30	`:lowerfirst` or `:upperfirst`

20	`:quotewildcard`
10	`:length`

Modifiers having the same precedence (i.e. listed on the same row in the above table) cannot be used together.

Variables are referenced within text strings using the construct '`${name}`', where *name* is the name of the variable as it appeared in the first parameter to the `set` statement. For example:

```
require "variables";

set "sender" "root
":

if envelope :matches "${sender}"
{
   ...
}
```

Match variables refer to parts of the most recently evaluated successful match of type `:matches` or `:regex`. They have names consisting entirely of decimal digits. The variable '`${0}`' refers to the entire matched expression. The variable '`${1}`' refers to the substring matching the first occurrence of the wildcard ('`?`' and '`*`'), '`${2}`' refers to the second occurrence and so on. The wildcards match as little as possible (non-greedy matching). For example:

```
require ["variables", "fileinto"];

if header :matches "List-ID" "*<*
" {
   fileinto "INBOX.lists.${2}";
   stop;
}
```

If `:regex` match is used, the match variables starting from '`${1}`' refer to the substrings of the argument value matching subsequent parenthesized groups of the regular expression.

`string` [*comparator*] [*match-type*] *source*(`string-list`) [Test]
keys(`string-list`)

The `string` test compares two strings according to the selected comparator and match type. The test evaluates to '`true`' if any two strings from *source* and *keys* match.

The '`:count`' match used in '`string`' counts each empty string as 0, and each non-empty one as 1. The count of a string list is the sum of the counts of the member strings.

5.8.4 environment

The '`environment`' extension complies with *RFC 5183*. It is a built-in extension. It introduces the following test:

`environment` [*comparator*] [*match-type*] *name*(`string`) [Test]
keys(`string-list`)

The `environment` test evaluates to '`true`' if the value of the environment items *name* matches any string from *keys*.

The following environment items are defined:

domain The primary DNS domain of the machine where the Sieve script is executing.

host The fully-qualified domain name of the host where the Sieve script is executing.

location Type of service that is evaluating the script. Depending on the utility that is evaluating the script it is:

Utility	**Location**
sieve	'`"MUA"`', or set with the `--environment` option.
maidag	'`"MDA"`'
inc	'`"MUA"`'

name The string '`GNU Mailutils`'

phase The point relative to final delivery where the Sieve script is being evaluated. Depending on the utility that is evaluating the script it is:

Utility	**Location**
sieve	'`post`' unless set with the `--environment` option.
maidag	'`"during"`'
inc	'`"post"`'

version Mailutils version string (e.g. '`3.19`').

5.8.5 The numaddr extension

This is an example loadable extension. Section 5.6.2 [External Tests], page 181.

5.8.6 The editheader extension

The `editheader` extension complies with *RFC 5293*. It provides the following actions:

`addheader` [`:last`] *field-name*(`string`) *value*(`string`) [Action]

Adds a header field to the existing message header. By default the header is inserted at the beginning of the header list. If the tag `:last` is specified, it is appended at the end.

`deleteheader"` [`:index` *fieldno*(`number`) `:last`] [*comparator*] [*match-type*] *field-name*(`string`) [*value-patterns*(`string-list`)] [Action]

Deletes occurrences of the header field matching the criteria.

The *value-patterns*, if specified, determines which occurrences of the header fielde to delete. If not supplied, *comparator* and *match-type* are silently ignored.

If '`:index` *fieldno*' is specified, only the numbered occurrence of the named header field will be matched (header numbering begins at 1), If `:last` is specified, the count

is backwards; 1 denotes the last named header field, 2 the second to last, and so on. The counting happens before the *value-patterns* match, if any. Thus, e.g. the action

```
deleteheader :index 1 :contains "Delivered-To" "bob@example.com";
```

would delete the first '`Delivered-To`' header field if it contains the string '`bob@example.com`'.

5.8.7 The list extension

Section 5.6.2 [External Tests], page 181.

5.8.8 The moderator extension

A loadable extension implementing a moderator robot for Mailman-driven mail archives. Section 5.7.2 [External Actions], page 188.

5.8.9 The pipe extension

A loadable extension for external command execution. It provides the `pipe` action (see Section 5.7.2 [External Actions], page 188) and test (see Section 5.6.2 [External Tests], page 181).

5.8.10 The spamd extension

Implements a test which interfaces to SpamAssassin filter. This is a loadable extension. see Section 5.6.2 [External Tests], page 181.

5.8.11 The timestamp extension

The loadable extension `timestamp` implements a test for comparing the value of a structured date header field with the given date.

Note: this extension will probably phase away in favor of the `date` Sieve extension (*RFC 5260*).

5.8.12 The vacation extension

The loadable extension `vacation` provides the action intended to inform the sender that the recipient is not currently reading his mail.

See Section 5.7.2 [External Actions], page 188.

5.9 GNU Extensions

This section summarizes the GNU extensions to the sieve language

1. Multiline strings syntax

   GNU libmu_sieve 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 5.4 [Require Statement], page 175).
   - 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. `fileinto` action

   The `fileinto` action allows to specify permissions on the mailbox, in case it will be created (see [fileinto], page 185).
7. 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.

6 Reporting Bugs

Email bug reports to `bug-mailutils@gnu.org`.

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`.

7 Getting News About GNU Mailutils

The two places to look for any news regarding GNU Mailutils are the Mailutils homepage at `http://mailutils.org` or `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://mailutils.org/manual`. See also Mailutils Wiki (`http://mailutils.org/wiki`) for the latest updates.

8 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

==========

Editor's note:

This node is to be written.

==========

Appendix B Date Input Formats

First, a quote:

> Our units of temporal measurement, from seconds on up to months, are so complicated, asymmetrical and disjunctive so as to make coherent mental reckoning in time all but impossible. Indeed, had some tyrannical god contrived to enslave our minds to time, to make it all but impossible for us to escape subjection to sodden routines and unpleasant surprises, he could hardly have done better than handing down our present system. It is like a set of trapezoidal building blocks, with no vertical or horizontal surfaces, like a language in which the simplest thought demands ornate constructions, useless particles and lengthy circumlocutions. Unlike the more successful patterns of language and science, which enable us to face experience boldly or at least level-headedly, our system of temporal calculation silently and persistently encourages our terror of time.
>
> ... It is as though architects had to measure length in feet, width in meters and height in ells; as though basic instruction manuals demanded a knowledge of five different languages. It is no wonder then that we often look into our own immediate past or future, last Tuesday or a week from Sunday, with feelings of helpless confusion. ...
>
> — Robert Grudin, *Time and the Art of Living*.

This section describes the textual date representations that GNU programs accept. These are the strings you, as a user, can supply as arguments to the various programs. The C interface (via the `get_date` function) is not described here.

B.1 General date syntax

A *date* is a string, possibly empty, containing many items separated by whitespace. The whitespace may be omitted when no ambiguity arises. The empty string means the beginning of today (i.e., midnight). Order of the items is immaterial. A date string may contain many flavors of items:

- calendar date items
- time of day items
- time zone items
- day of the week items
- relative items
- pure numbers.

We describe each of these item types in turn, below.

A few ordinal numbers may be written out in words in some contexts. This is most useful for specifying day of the week items or relative items (see below). Among the most commonly used ordinal numbers, the word '`last`' stands for −1, '`this`' stands for 0, and '`first`' and '`next`' both stand for 1. Because the word '`second`' stands for the unit of time there is no way to write the ordinal number 2, but for convenience '`third`' stands for 3, '`fourth`' for 4, '`fifth`' for 5, '`sixth`' for 6, '`seventh`' for 7, '`eighth`' for 8, '`ninth`' for 9, '`tenth`' for 10, '`eleventh`' for 11 and '`twelfth`' for 12.

When a month is written this way, it is still considered to be written numerically, instead of being "spelled in full"; this changes the allowed strings.

In the current implementation, only English is supported for words and abbreviations like 'AM', 'DST', 'EST', 'first', 'January', 'Sunday', 'tomorrow', and 'year'.

The output of the date command is not always acceptable as a date string, not only because of the language problem, but also because there is no standard meaning for time zone items like 'IST'. When using date to generate a date string intended to be parsed later, specify a date format that is independent of language and that does not use time zone items other than 'UTC' and 'Z'. Here are some ways to do this:

```
$ LC_ALL=C TZ=UTC0 date
Mon Mar  1 00:21:42 UTC 2004
$ TZ=UTC0 date +'%Y-%m-%d %H:%M:%SZ'
2004-03-01 00:21:42Z
$ date --iso-8601=ns | tr T ' '  # --iso-8601 is a GNU extension.
2004-02-29 16:21:42,692722128-0800
$ date --rfc-2822  # a GNU extension
Sun, 29 Feb 2004 16:21:42 -0800
$ date +'%Y-%m-%d %H:%M:%S %z'  # %z is a GNU extension.
2004-02-29 16:21:42 -0800
$ date +'@%s.%N'  # %s and %N are GNU extensions.
@1078100502.692722128
```

Alphabetic case is completely ignored in dates. Comments may be introduced between round parentheses, as long as included parentheses are properly nested. Hyphens not followed by a digit are currently ignored. Leading zeros on numbers are ignored.

Invalid dates like '2005-02-29' or times like '24:00' are rejected. In the typical case of a host that does not support leap seconds, a time like '23:59:60' is rejected even if it corresponds to a valid leap second.

B.2 Calendar date items

A *calendar date item* specifies a day of the year. It is specified differently, depending on whether the month is specified numerically or literally. All these strings specify the same calendar date:

```
1972-09-24     # ISO 8601.
72-9-24        # Assume 19xx for 69 through 99,
               # 20xx for 00 through 68.
72-09-24       # Leading zeros are ignored.
9/24/72        # Common U.S. writing.
24 September 1972
24 Sept 72     # September has a special abbreviation.
24 Sep 72      # Three-letter abbreviations always allowed.
Sep 24, 1972
24-sep-72
24sep72
```

The year can also be omitted. In this case, the last specified year is used, or the current year if none. For example:

```
9/24
sep 24
```

Here are the rules.

For numeric months, the ISO 8601 format '*year*`-`*month*`-`*day*' is allowed, where *year* is any positive number, *month* is a number between 01 and 12, and *day* is a number between 01 and 31. A leading zero must be present if a number is less than ten. If *year* is 68 or smaller, then 2000 is added to it; otherwise, if *year* is less than 100, then 1900 is added to it. The construct '*month*`/`*day*`/`*year*', popular in the United States, is accepted. Also '*month*`/`*day*', omitting the year.

Literal months may be spelled out in full: '`January`', '`February`', '`March`', '`April`', '`May`', '`June`', '`July`', '`August`', '`September`', '`October`', '`November`' or '`December`'. Literal months may be abbreviated to their first three letters, possibly followed by an abbreviating dot. It is also permitted to write '`Sept`' instead of '`September`'.

When months are written literally, the calendar date may be given as any of the following:

```
day month year
day month
month day year
day-month-year
```

Or, omitting the year:

```
month day
```

B.3 Time of day items

A *time of day item* in date strings specifies the time on a given day. Here are some examples, all of which represent the same time:

```
20:02:00.000000
20:02
8:02pm
20:02-0500      # In EST (U.S.  Eastern Standard Time).
```

More generally, the time of day may be given as '*hour*`:`*minute*`:`*second*', where *hour* is a number between 0 and 23, *minute* is a number between 0 and 59, and *second* is a number between 0 and 59 possibly followed by '`.`' or '`,`' and a fraction containing one or more digits. Alternatively, '`:`*second*' can be omitted, in which case it is taken to be zero. On the rare hosts that support leap seconds, *second* may be 60.

If the time is followed by '`am`' or '`pm`' (or '`a.m.`' or '`p.m.`'), *hour* is restricted to run from 1 to 12, and '`:`*minute*' may be omitted (taken to be zero). '`am`' indicates the first half of the day, '`pm`' indicates the second half of the day. In this notation, 12 is the predecessor of 1: midnight is '`12am`' while noon is '`12pm`'. (This is the zero-oriented interpretation of '`12am`' and '`12pm`', as opposed to the old tradition derived from Latin which uses '`12m`' for noon and '`12pm`' for midnight.)

The time may alternatively be followed by a time zone correction, expressed as '*shhmm*', where *s* is '`+`' or '`-`', *hh* is a number of zone hours and *mm* is a number of zone minutes. The zone minutes term, *mm*, may be omitted, in which case the one- or two-digit correction is interpreted as a number of hours. You can also separate *hh* from *mm* with a colon. When a time zone correction is given this way, it forces interpretation of the time relative to

Coordinated Universal Time (UTC), overriding any previous specification for the time zone or the local time zone. For example, '`+0530`' and '`+05:30`' both stand for the time zone 5.5 hours ahead of UTC (e.g., India). This is the best way to specify a time zone correction by fractional parts of an hour. The maximum zone correction is 24 hours.

Either '`am`'/'`pm`' or a time zone correction may be specified, but not both.

B.4 Time zone items

A *time zone item* specifies an international time zone, indicated by a small set of letters, e.g., '`UTC`' or '`Z`' for Coordinated Universal Time. Any included periods are ignored. By following a non-daylight-saving time zone by the string '`DST`' in a separate word (that is, separated by some white space), the corresponding daylight saving time zone may be specified. Alternatively, a non-daylight-saving time zone can be followed by a time zone correction, to add the two values. This is normally done only for '`UTC`'; for example, '`UTC+05:30`' is equivalent to '`+05:30`'.

Time zone items other than '`UTC`' and '`Z`' are obsolescent and are not recommended, because they are ambiguous; for example, '`EST`' has a different meaning in Australia than in the United States. Instead, it's better to use unambiguous numeric time zone corrections like '`-0500`', as described in the previous section.

If neither a time zone item nor a time zone correction is supplied, time stamps are interpreted using the rules of the default time zone (see Section B.9 [Specifying time zone rules], page 210).

B.5 Day of week items

The explicit mention of a day of the week will forward the date (only if necessary) to reach that day of the week in the future.

Days of the week may be spelled out in full: '`Sunday`', '`Monday`', '`Tuesday`', '`Wednesday`', '`Thursday`', '`Friday`' or '`Saturday`'. Days may be abbreviated to their first three letters, optionally followed by a period. The special abbreviations '`Tues`' for '`Tuesday`', '`Wednes`' for '`Wednesday`' and '`Thur`' or '`Thurs`' for '`Thursday`' are also allowed.

A number may precede a day of the week item to move forward supplementary weeks. It is best used in expression like '`third monday`'. In this context, '`last` *day*' or '`next` *day*' is also acceptable; they move one week before or after the day that *day* by itself would represent.

A comma following a day of the week item is ignored.

B.6 Relative items in date strings

Relative items adjust a date (or the current date if none) forward or backward. The effects of relative items accumulate. Here are some examples:

```
1 year
1 year ago
3 years
2 days
```

The unit of time displacement may be selected by the string '`year`' or '`month`' for moving by whole years or months. These are fuzzy units, as years and months are not all of equal

duration. More precise units are '`fortnight`' which is worth 14 days, '`week`' worth 7 days, '`day`' worth 24 hours, '`hour`' worth 60 minutes, '`minute`' or '`min`' worth 60 seconds, and '`second`' or '`sec`' worth one second. An '`s`' suffix on these units is accepted and ignored.

The unit of time may be preceded by a multiplier, given as an optionally signed number. Unsigned numbers are taken as positively signed. No number at all implies 1 for a multiplier. Following a relative item by the string '`ago`' is equivalent to preceding the unit by a multiplier with value -1.

The string '`tomorrow`' is worth one day in the future (equivalent to '`day`'), the string '`yesterday`' is worth one day in the past (equivalent to '`day ago`').

The strings '`now`' or '`today`' are relative items corresponding to zero-valued time displacement, these strings come from the fact a zero-valued time displacement represents the current time when not otherwise changed by previous items. They may be used to stress other items, like in '`12:00 today`'. The string '`this`' also has the meaning of a zero-valued time displacement, but is preferred in date strings like '`this thursday`'.

When a relative item causes the resulting date to cross a boundary where the clocks were adjusted, typically for daylight saving time, the resulting date and time are adjusted accordingly.

The fuzz in units can cause problems with relative items. For example, '`2003-07-31 -1 month`' might evaluate to 2003-07-01, because 2003-06-31 is an invalid date. To determine the previous month more reliably, you can ask for the month before the 15th of the current month. For example:

```
$ date -R
Thu, 31 Jul 2003 13:02:39 -0700
$ date --date='-1 month' +'Last month was %B?'
Last month was July?
$ date --date="$(date +%Y-%m-15) -1 month" +'Last month was %B!'
Last month was June!
```

Also, take care when manipulating dates around clock changes such as daylight saving leaps. In a few cases these have added or subtracted as much as 24 hours from the clock, so it is often wise to adopt universal time by setting the `TZ` environment variable to '`UTC0`' before embarking on calendrical calculations.

B.7 Pure numbers in date strings

The precise interpretation of a pure decimal number depends on the context in the date string.

If the decimal number is of the form *yyyymmdd* and no other calendar date item (see Section B.2 [Calendar date items], page 206) appears before it in the date string, then *yyyy* is read as the year, *mm* as the month number and *dd* as the day of the month, for the specified calendar date.

If the decimal number is of the form *hhmm* and no other time of day item appears before it in the date string, then *hh* is read as the hour of the day and *mm* as the minute of the hour, for the specified time of day. *mm* can also be omitted.

If both a calendar date and a time of day appear to the left of a number in the date string, but no relative item, then the number overrides the year.

B.8 Seconds since the Epoch

If you precede a number with '`@`', it represents an internal time stamp as a count of seconds. The number can contain an internal decimal point (either '`.`' or '`,`'); any excess precision not supported by the internal representation is truncated toward minus infinity. Such a number cannot be combined with any other date item, as it specifies a complete time stamp.

Internally, computer times are represented as a count of seconds since an epoch—a well-defined point of time. On GNU and POSIX systems, the epoch is 1970-01-01 00:00:00 UTC, so '`@0`' represents this time, '`@1`' represents 1970-01-01 00:00:01 UTC, and so forth. GNU and most other POSIX-compliant systems support such times as an extension to POSIX, using negative counts, so that '`@-1`' represents 1969-12-31 23:59:59 UTC.

Traditional Unix systems count seconds with 32-bit two's-complement integers and can represent times from 1901-12-13 20:45:52 through 2038-01-19 03:14:07 UTC. More modern systems use 64-bit counts of seconds with nanosecond subcounts, and can represent all the times in the known lifetime of the universe to a resolution of 1 nanosecond.

On most hosts, these counts ignore the presence of leap seconds. For example, on most hosts '`@915148799`' represents 1998-12-31 23:59:59 UTC, '`@915148800`' represents 1999-01-01 00:00:00 UTC, and there is no way to represent the intervening leap second 1998-12-31 23:59:60 UTC.

B.9 Specifying time zone rules

Normally, dates are interpreted using the rules of the current time zone, which in turn are specified by the `TZ` environment variable, or by a system default if `TZ` is not set. To specify a different set of default time zone rules that apply just to one date, start the date with a string of the form '`TZ="`*rule*`"`'. The two quote characters ('`"`') must be present in the date, and any quotes or backslashes within *rule* must be escaped by a backslash.

For example, with the GNU `date` command you can answer the question "What time is it in New York when a Paris clock shows 6:30am on October 31, 2004?" by using a date beginning with '`TZ="Europe/Paris"`' as shown in the following shell transcript:

```
$ export TZ="America/New_York"
$ date --date='TZ="Europe/Paris" 2004-10-31 06:30'
Sun Oct 31 01:30:00 EDT 2004
```

In this example, the `--date` operand begins with its own `TZ` setting, so the rest of that operand is processed according to '`Europe/Paris`' rules, treating the string '`2004-10-31 06:30`' as if it were in Paris. However, since the output of the `date` command is processed according to the overall time zone rules, it uses New York time. (Paris was normally six hours ahead of New York in 2004, but this example refers to a brief Halloween period when the gap was five hours.)

A `TZ` value is a rule that typically names a location in the '`tz`' database (`http://www.twinsun.com/tz/tz-link.htm`). A recent catalog of location names appears in the TWiki Date and Time Gateway (`http://twiki.org/cgi-bin/xtra/tzdate`). A few non-GNU hosts require a colon before a location name in a `TZ` setting, e.g., '`TZ=":America/New_York"`'.

The '`tz`' database includes a wide variety of locations ranging from '`Arctic/Longyearbyen`' to '`Antarctica/South_Pole`', but if you are at sea and

have your own private time zone, or if you are using a non-GNU host that does not support the '`tz`' database, you may need to use a POSIX rule instead. Simple POSIX rules like '`UTC0`' specify a time zone without daylight saving time; other rules can specify simple daylight saving regimes. See Section "Specifying the Time Zone with `TZ`" in *The GNU C Library*.

B.10 Authors of `get_date`

`get_date` was originally implemented by Steven M. Bellovin (`smb@research.att.com`) while at the University of North Carolina at Chapel Hill. The code was later tweaked by a couple of people on Usenet, then completely overhauled by Rich $alz (`rsalz@bbn.com`) and Jim Berets (`jberets@bbn.com`) in August, 1990. Various revisions for the GNU system were made by David MacKenzie, Jim Meyering, Paul Eggert and others.

This chapter was originally produced by François Pinard (`pinard@iro.umontreal.ca`) from the `getdate.y` source code, and then edited by K. Berry (`kb@cs.umb.edu`).

Appendix C Date/time Format String

This appendix documents the format specifications for outputting date/time values. It is used, in particular, by the `mail` utility (see [headline], page 80).

Essentially, it is a reproduction of the man page for GNU `strftime` function. Some of the conversion specifiers might not be available on all systems, due to differences in '`strftime`' between systems. If unsure, please consult Section "strftime" in *the strftime(3) man page*.

Ordinary characters placed in the format string are reproduced without conversion. Conversion specifiers are introduced by a '`%`' character, and are replaced as follows:

%a — The abbreviated weekday name according to the current locale.

%A — The full weekday name according to the current locale.

%b — The abbreviated month name according to the current locale.

%B — The full month name according to the current locale.

%c — The preferred date and time representation for the current locale.

%C — The century number (year/100) as a 2-digit integer.

%d — The day of the month as a decimal number (range 01 to 31).

%D — Equivalent to '`%m/%d/%y`'.

%e — Like '`%d`', the day of the month as a decimal number, but a leading zero is replaced by a space.

%E — Modifier: use alternative format, see below (see [conversion specs], page 215).

%F — Equivalent to '`%Y-%m-%d`' (the ISO 8601 date format).

%G — The ISO 8601 year with century as a decimal number. The 4-digit year corresponding to the ISO week number (see '`%V`'). This has the same format and value as '`%y`', except that if the ISO week number belongs to the previous or next year, that year is used instead.

%g — Like '`%G`', but without century, i.e., with a 2-digit year (00-99).

%h	Equivalent to '`%b`'.
%H	The hour as a decimal number using a 24-hour clock (range 00 to 23).
%I	The hour as a decimal number using a 12-hour clock (range 01 to 12).
%j	The day of the year as a decimal number (range 001 to 366).
%k	The hour (24-hour clock) as a decimal number (range 0 to 23); single digits are preceded by a blank. (See also '`%H`'.)
%l	The hour (12-hour clock) as a decimal number (range 1 to 12); single digits are preceded by a blank. (See also '`%I`'.)
%m	The month as a decimal number (range 01 to 12).
%M	The minute as a decimal number (range 00 to 59).
%n	A newline character.
%O	Modifier: use alternative format, see below (see [conversion specs], page 215).
%p	Either '`AM`' or '`PM`' according to the given time value, or the corresponding strings for the current locale. Noon is treated as '`pm`' and midnight as '`am`'.
%P	Like '`%p`' but in lowercase: '`am`' or '`pm`' or a corresponding string for the current locale.
%r	The time in '`a.m.`' or '`p.m.`' notation. In the POSIX locale this is equivalent to '`%I:%M:%S %p`'.
%R	The time in 24-hour notation ('`%H:%M`'). For a version including the seconds, see '`%T`' below.
%s	The number of seconds since the Epoch, i.e., since 1970-01-01 00:00:00 UTC.
%S	The second as a decimal number (range 00 to 61).
%t	A tab character.

%T	The time in 24-hour notation ('`%H:%M:%S`').
%u	The day of the week as a decimal, range 1 to 7, Monday being 1. See also '`%w`'.
%U	The week number of the current year as a decimal number, range 00 to 53, starting with the first Sunday as the first day of week 01. See also '`%V`' and '`%W`'.
%V	The ISO 8601:1988 week number of the current year as a decimal number, range 01 to 53, where week 1 is the first week that has at least 4 days in the current year, and with Monday as the first day of the week. See also '`%U`' and '`%W`'.
%w	The day of the week as a decimal, range 0 to 6, Sunday being 0. See also '`%u`'.
%W	The week number of the current year as a decimal number, range 00 to 53, starting with the first Monday as the first day of week 01.
%x	The preferred date representation for the current locale without the time.
%X	The preferred time representation for the current locale without the date.
%y	The year as a decimal number without a century (range 00 to 99).
%Y	The year as a decimal number including the century.
%z	The time-zone as hour offset from GMT. Required to emit RFC822-conformant dates (using '`%a, %d %b %Y %H:%M:%S %z`')
%Z	The time zone or name or abbreviation.
%+	The date and time in *date(1)* format.
%%	A literal '`%`' character.

Some conversion specifiers can be modified by preceding them by the '`E`' or '`O`' modifier to indicate that an alternative format should be used. If the alternative format or specification does not exist for the current locale, the behaviour will be as if the unmodified conversion specification were used. The Single Unix Specification mentions '`%Ec`', '`%EC`',

‘%Ex’, ‘%EX’, ‘%Ry’, ‘%EY’, ‘%Od’, ‘%Oe’, ‘%OH’, ‘%OI’, ‘%Om’, ‘%OM’, ‘%OS’, ‘%Ou’, ‘%OU’, ‘%OV’, ‘%Ow’, ‘%OW’, ‘%Oy’, where the effect of the ‘O’ modifier is to use alternative numeric symbols (say, roman numerals), and that of the ‘E’ modifier is to use a locale-dependent alternative representation.

Appendix D Configuring Help Summary

Running *prog* `--help` displays the short usage summary for *prog* utility (see Section 3.1.2 [Common Options], page 8). This summary is organized by *groups* of semantically close options. The options within each group are printed in the following order: a short option, eventually followed by a list of corresponding long option names, followed by a short description of the option. For example, here is an excerpt from the actual *sieve* `--help` output:

```
-c, --compile-only          Compile script and exit
-d, --debug[=FLAGS]         Debug flags
-e, --email=ADDRESS         Override user email address
```

The exact visual representation of the help output is configurable via `ARGP_HELP_FMT` environment variable. The value of this variable is a comma-separated list of *format variable* assignments. There are two kinds of format variables. An *offset variable* keeps the offset of some part of help output text from the leftmost column on the screen. A *boolean* variable is a flag that toggles some output feature on or off. Depending on the type of the corresponding variable, there are two kinds of assignments:

Offset assignment
: The assignment to an offset variable has the following syntax:

  ```
  variable=value
  ```

  where *variable* is the variable name, and *value* is a numeric value to be assigned to the variable.

Boolean assignment
: To assign `true` value to a variable, simply put this variable name. To assign `false` value, prefix the variable name with '`no-`'. For example:

  ```
  # Assign true value:
  dup-args
  # Assign false value:
  no-dup-args
  ```

Following variables are declared:

`boolean dup-args` [Help Output]
: If true, arguments for an option are shown with both short and long options, even when a given option has both forms, for example:

  ```
  -e ADDRESS, --email=ADDRESS        Override user email address
  ```

  If false, then if an option has both short and long forms, the argument is only shown with the long one, for example:

  ```
  -e, --email=ADDRESS        Override user email address
  ```

  and a message indicating that the argument is applicable to both forms is printed below the options. This message can be disabled using `dup-args-note` (see below).

  The default is false.

`boolean dup-args-note` [Help Output]
: If this variable is true, which is the default, the following notice is displayed at the end of the help output:

Mandatory or optional arguments to long options are also mandatory or optional for any corresponding short options.

Setting `no-dup-args-note` inhibits this message. Normally, only one of variables `dup-args` or `dup-args-note` should be set.

`offset short-opt-col` [Help Output]

Column in which short options start. Default is 2.

```
$ sieve --help|grep ADDRESS
  -e, --email=ADDRESS        Override user email address
$ ARGP_HELP_FMT=short-opt-col=6 sieve --help|grep ARCHIVE
      -e, --email=ADDRESS        Override user email address
```

`offset long-opt-col` [Help Output]

Column in which long options start. Default is 6. For example:

```
$ sieve --help|grep ADDRESS
  -e, --email=ADDRESS        Override user email address
$ ARGP_HELP_FMT=long-opt-col=16 sieve --help|grep ADDRESS
  -e,           --email=ADDRESS        Override user email address
```

`offset doc-opt-col` [Help Output]

Column in which *doc options* start. A doc option isn't actually an option, but rather an arbitrary piece of documentation that is displayed in much the same manner as the options. For example, in the output of `folder --help`:

```
Usage: folder [OPTION...] [action] [msg]
GNU MH folder
  Actions are:
      --list                 List the contents of the folder stack
  ...
```

the string '`Actions are:`' is a doc option. Thus, if you set *`ARGP_HELP_FMT=doc-opt-col=6`* the above part of the help output will look as follows:

```
Usage: folder [OPTION...] [action] [msg]
GNU MH folder
      Actions are:
      --list                 List the contents of the folder stack
  ...
```

`offset opt-doc-col` [Help Output]

Column in which option description starts. Default is 29.

```
$ sieve --help|grep ADDRESS
  -e, --email=ADDRESS        Override user email address
$ ARGP_HELP_FMT=opt-doc-col=19 sieve --help|grep ADDRESS
  -e, --email=ADDRESS   Override user email address
$ ARGP_HELP_FMT=opt-doc-col=9 sieve --help|grep -i ADDRESS
  -e, --email=ADDRESS
         Override user email address
```

Notice, that the description starts on a separate line if `opt-doc-col` value is too small.

`offset header-col` [Help Output]

Column in which *group headers* are printed. A group header is a descriptive text preceding an option group. For example, in the following text:

```
Sieve options
 -I, --includedir=DIR       Append directory DIR to the
                            list of include directories
```

the text '`Sieve options`' is a group header.

The default value is 1.

`offset usage-indent` [Help Output]

Indentation of wrapped usage lines. Affects `--usage` output. Default is 12.

`offset rmargin` [Help Output]

Right margin of the text output. Used for wrapping.

Appendix E 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.

E.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.

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Variable Index

R

S

T

U

V

X

Keyword Index

L

M

N

O

P

R

S

T

U

V

Program Index

Concept Index

This is a general index of all issues discussed in this manual

~

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V