mailutils, SDK.

version 0.0.9b, 14 October 2001

Alain Magloire et al.

Published by the Free Software Foundation,
59 Temple Place - Suite 330,
Boston, MA 02111-1307, USA

Short Contents

1 Introduction

GNU Mailutils offers a general purpose library whose aim is to provide a rich set of functions for accessing different mailbox formats and mailers.

1.1 References

For more information on,

- SMTP
  - *RFC 2821: Simple Mail Transfer Protocol*
  - *RFC 2368: The mailto URL scheme*
- POP3
  - *RFC 1939: Post Office Protocol - Version 3*
  - *RFC 1734: POP3 AUTHentication command*
  - *RFC 1957: Some Observations on Implementations of the Post Office Protocol (POP3)*
  - *RFC 2449: POP3 Extension Mechanism*
  - *RFC 2384: POP URL Scheme*
- IMAP4
  - *RFC 2060: INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*
  - *RFC 2088: IMAP4 non-synchronizing literals*
  - *RFC 2193: IMAP4 Mailbox Referrals*
  - *RFC 2221: IMAP4 Login Referrals*
  - *RFC 2342: IMAP4 Namespace*
  - *RFC 2192: IMAP URL Scheme*
  - *RFC 1731: IMAP4 Authentication Mechanisms*
  - *RFC 2245: Anonymous SASL Mechanism*
- message formats
  - *RFC 2822: Internet Message Format*
  - *RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*
  - *RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*
  - *RFC 2047: Multipurpose Internet Mail Extensions (MIME) Part Three: Message Header Extensions for Non-ASCII Text*
  - *RFC 2049: Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples*
  - *RFC 2111: Content-ID and Message-ID Uniform Resource Locators*
- miscellaneous related topics
  - *RFC 3028: Sieve: A Mail Filtering Language*
  - *RFC 2298: An Extensible Message Format for Message Disposition Notifications*

- *RFC 1738: Uniform Resource Locators (URL)*
- *Internet Email Protocols: A Developer's Guide, by Kevin Johnson*

2 Concrete API

2.1 POP3

```
/* Prefix pop3_ is reserved */
#include <mailutils/pop3.h>
```

The purpose of the Post Office Protocol Version 3 (POP3) is to permit a client to download a maildrop from a remote server. It does not provide complex or extensive operation on the maildrop. When the client successfully authenticates, the server acquires an exclusive access lock on the mailbox and holds it the entire duration of the session. After the authentication, the session enters transaction state and the client may issues commands to access messages in the mailbox.

- Authorization state
  - User
  - Pass
  - Quit
  - Capa (extension)
  - Auth (extension)
- Transaction state
  - Stat
  - List
  - Retr
  - Dele
  - Noop
  - Rset
  - Top
  - Uidl (extension)
  - Quit
  - Capa (extension)

When the command Quit is issue the session enters the update state. The servers removes all messages marked deleted, releases the exclusive lock and closes the TCP connection.

2.1.1 Commands

An opaque structure *pop3_t* is use as a handle for the session, it is allocated and initialized by calling `pop3_create ()`. All Functions will wait for a reply from the POP3 server before returning. The duration of the wait can be set by calling `pop3_set_timeout ()`, the default is 10 minutes[1]. Once a successful connection is established with `pop3_connect ()`, two builtins authentications are provided `pop3_apop ()` or `pop3_user ()`/`pop3_pass ()`. The `pop3_stat ()` and `pop3_list ()` functions can be use to get the number and size of messages. Downloading of messages is done via a stream provided by `pop3_retr ()` or `pop3_top ()`[2]. The `stream_t` should be destroyed to indicate to the library that the action is finished. POP3 only provide a single channel for operation, it means only one operation can be done at a time, all the functions will return MU_ERROR_OPERATION_IN_PROGRESS if call during another operation. The functions `pop3_list_all ()`, `pop3_uidl_all ()` and `pop3_capa ()` return iterators `pop3_list_current ()`, `pop3_uidl_current ()` are provided as cover function to format the string return by `iterator_current ()`, `iterator_destroy ()` must be call to release any resources.

In a multithreaded application, only one thread should access *pop3_t* handles.

2.1.1.1 Initialization

`int` **pop3_create** (`pop3_t *` *pop3*) Function

Allocate and initialize a *pop3* handle.

`MU_ERROR_NO_MEMORY`

`MU_ERROR_INVALID_PARAMETER`

`void` **pop3_destroy** (`pop3_t *`*pop3*) Function

When a POP3 session is finished, the structure must be `free ()`'ed to reclaim memory.

`int` **pop3_connect** (`pop3_t, const char *`*host*`, unsigned` *port*`, int` *flags*) Function

A connection is established by calling `pop3d_open ()`, the previous connection is close first. If non-blocking the function should be recalled until the return value is not MU_ERROR_TRY_AGAIN or MU_ERROR_IN_PROGRESS.

`MU_ERROR_NO_MEMORY`

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_IO`

`MU_ERROR_IN_PROGRESS`

`MU_ERROR_TRY_AGAIN`

`MU_ERROR_OPERATION_DENIED`

`MU_ERROR_TIMEOUT`

`MU_ERROR_NO_LOCK`

---

[1] **Caution:** Although the *RFC 1939* specifies that the minimum default timeout is ten minutes many servers has shorter idle period, care should be taken to at least send a `pop3_noop ()` between lengthy period of times.

[2] **Caution:** Some Internet Service Providers do not permit to leave mail on server and the message will be deleted once downloaded.

2.1.1.2 Carrier

`int` **pop3_set_carrier** (`pop3_t, stream_t` *carrier*); Function

The type of stream use to contact as server will be set to *carrier* in the *pop3_t* handle. Any previous *carrier* stream in the handle, will be close and release.

`MU_ERROR_INVALID_PARAMETER`

`int` **pop3_get_carrier** (`pop3_t, stream_t *`*carrier*); Function

Return the *pop3_t* carrier. If none was set, a new tcp stream will be created.

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_NO_MEMORY`

2.1.1.3 Apop

`int` **pop3_apop** (`pop3_t, const char *`*user*`, const char *`*secret*) Function

Apop offers an alternative to User/Pass authentication. For intermittent use of POP3, like checking for new mail, it is the preferred the authentication. It reduces the risk of password capture. The *user* and the shared *secret* are pass to `pop3_apop ()`, the MD5 digest is calculated by applying the times tamp given by the server in the greeting followed by the *secret*.

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_IO`

`MU_ERROR_TIMEOUT`

`MU_ERROR_TRY_AGAIN`

`MU_ERROR_OPERATION_DENIED`

2.1.1.4 Capa

`int` **pop3_capa** (`pop3_t, pop3_capa_iterator_t *`*iterator*) Function

The Capa command is send to the sever and the list of capabilities is return in an *iterator*. `iterator_current ()` gives an allocated string that should be `free ()`'ed. **Caution:** The iterator must be destroy after use, it will discard any remaining responses from CAPA and clear the way for new operations.

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_IO`

`MU_ERROR_TIMEOUT`

`MU_ERROR_TRY_AGAIN`

`MU_ERROR_OPERATION_DENIED`

```
#include <stdio.h>
#include <stdlib.h>
#include <mailutils/pop3.h>
```

```
void print_capabilities (pop3_t pop3)
{
   iterator_t iterator;
   status = pop3_capa (pop3, &iterator);
   if (status == 0)
    {
        for (iterator_first (iterator);
             !iterator_is_done (iterator);
             iterator_next (iterator))
          {
              char *capa;
              if (iterator_current (iterator, &capa) == 0)
               {;
                   printf ("CAPA: %s\n", capa);
                   free (capa);
               }
          }
        pop3_capa_destroy (&iterator);
    }
   else
     printf ("NONE\n");
}
```

2.1.1.5 Dele

int **pop3_dele** (pop3_t, unsigned *msgno*) Function

Sends a Dele to the servers who will mark the *msgno* for deletion. The *msgno* may not refer to a message already marked deleted. If successful any future reference to *msgno* in other operations will be an error, unless unmarked by RSET.

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_IO`

`MU_ERROR_TIMEOUT`

`MU_ERROR_TRY_AGAIN`

`MU_ERROR_OPERATION_DENIED`

2.1.1.6 List

int **pop3_list** (pop3_t, unsigned *msgno*, size_t **size*) Function

Sends a List for a specific *msgno* and returns the *size*.

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_IO`

`MU_ERROR_TIMEOUT`

`MU_ERROR_TRY_AGAIN`

`MU_ERROR_OPERATION_DENIED`

int **pop3_list_all** (pop3_t, iterator_t **iterator*) Function

Sends A List with no argument to the server. The *iterator* must be destroy after use, it will discard any remaining response from LIST and clear the way for new operations. A cover function `pop3_list_current ()` around to scan properly the string return by the `iterator_current ()`.

```
#include <stdio.h>
#include <stdlib.h>
#include <mailutils/pop3.h>

void print_list (pop3_t pop3)
{
   iterator_t iterator;
   status = pop3_list_all (pop3, &iterator);
   if (status == 0)
    {
        for (iterator_first (iterator);
             !iterator_is_done (iterator);
             iterator_next (iterator))
          {
              unsigned int msgno, size;
              if (pop3_list_current (iterator, &msgno, &size) == 0)
               {
                   printf ("LIST: %d %d\n", msgno, size);
               }
          }
        iterator (&iterator);
    }
   else
     printf ("NONE\n");
}
```

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_IO`

`MU_ERROR_TIMEOUT`

`MU_ERROR_TRY_AGAIN`

`MU_ERROR_OPERATION_DENIED`

int **pop3_list_current** (pop3_t, unsigned int **msgno*, size_t **size*) Function

Cover functions around `iterator_current ()` from an iterator created by `pop3_list_all ()` to format the result.

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_IO`

`MU_ERROR_TIMEOUT`

`MU_ERROR_TRY_AGAIN`

`MU_ERROR_OPERATION_DENIED`

2.1.1.7 Noop

int **pop3_noop** (pop3_t) Function

Sends a NOOP, useful to reset the timeout on the server.

MU_ERROR_INVALID_PARAMETER

MU_ERROR_IO

MU_ERROR_TIMEOUT

MU_ERROR_TRY_AGAIN

MU_ERROR_OPERATION_DENIED

2.1.1.8 Pass

int **pop3_pass** (pop3_t, const char **passwd*) Function

Sends the PASS, to authenticate after the USER command.

MU_ERROR_INVALID_PARAMETER

MU_ERROR_IO

MU_ERROR_TIMEOUT

MU_ERROR_TRY_AGAIN

MU_ERROR_OPERATION_DENIED

2.1.1.9 Quit

int **pop3_quit** (pop3_t) Function

Enter the UPDATE state. The server will delete all messages marked deleted before closing the connection.

MU_ERROR_INVALID_PARAMETER

MU_ERROR_IO

MU_ERROR_TIMEOUT

MU_ERROR_TRY_AGAIN

MU_ERROR_OPERATION_DENIED

2.1.1.10 Retr

int **pop3_retr** (pop3_t, unsigned *msgno*, stream_t *) Function

If successful a `stream_t` is created to allow downloading of the message, byte-stuff lines or handle internally, CRLFs are converted to LF. All other operations will fail until the stream is destroyed by the caller.

```
#include <stdio.h>
#include <mailutils/pop3.h>

int
```

```
print_message (pop3_t pop3, unsigned int msgno)
{
   stream_t stream;
   int status = pop3_retr (pop3, msgno, &stream);
   if (status == 0)
    {
       size_t n = 0;
       char buf[128];
       while ((stream_readline (stream, buf, sizeof buf, &n) == 0)
              && n)
         printf ("%s", buf);
       stream_release (stream);
    }
   return status;
}
```

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_IO`

`MU_ERROR_TIMEOUT`

`MU_ERROR_TRY_AGAIN`

`MU_ERROR_OPERATION_DENIED`

2.1.1.11 Rset

int **pop3_rset** (pop3_t) Function

Sends a RSET to unmark the messages marked as deleted.

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_IO`

`MU_ERROR_TIMEOUT`

`MU_ERROR_TRY_AGAIN`

`MU_ERROR_OPERATION_DENIED`

2.1.1.12 Stat

int **pop3_stat** (pop3_t, unsigned *msgno*, unsigned **msg_count*, size_t **size*) Function

The number of messages in the mailbox and the size of the mailbox in octets. **Caution:** The size is in RFC822 where line termination is CRLF, messages marked as deleted are not counted in either total.

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_IO`

`MU_ERROR_TIMEOUT`

`MU_ERROR_TRY_AGAIN`

`MU_ERROR_OPERATION_DENIED`

2.1.1.13 Top

int **pop3_top** (pop3_t, unsigned int *msgno*, unsigned int *lines*, stream_t **stream*) Function

If successful a stream is created to allow downloading of the header, byte-stuff lines or handle internally, CRLFs are converted to LF. All other operations will failed until the stream is destroyed by the caller.

```
#include <stdio.h>
#include <mailutils/pop3.h>

int
print_top (pop3_t pop3, unsigned int msgno, unsigned int lines)
{
    stream_t stream;
    int status = pop3_top (pop3, msgno, lines, &stream);
    if (status == 0)
     {
        size_t n = 0;
        char buf[128];
        while ((stream_readline (stream, buf, sizeof buf, &n) == 0)
               && n)
          printf ("%s", buf);
        stream_release (stream);
     }
  return status;
}
```

MU_ERROR_INVALID_PARAMETER

MU_ERROR_IO

MU_ERROR_TIMEOUT

MU_ERROR_TRY_AGAIN

MU_ERROR_OPERATION_DENIED

2.1.1.14 Uidl

int **pop3_uidl** (pop3_t, unsigned int *msgno*, char ***uidl*) Function

The Uniq Identifier is return in *uidl*, the string must be free ()'ed, by the caller.

MU_ERROR_INVALID_PARAMETER

MU_ERROR_IO

MU_ERROR_TIMEOUT

MU_ERROR_TRY_AGAIN

MU_ERROR_OPERATION_DENIED

int **pop3_uidl_all** (pop3_t, iterator_t * *iterator*) Function

An iterator object is return to iterate through the response and must be destroyed by the caller.

```
#include <stdio.h>
#include <stdlib.h>
#include <mailutils/pop3.h>

void print_uidl (pop3_t pop3)
{
   iterator_t iterator;
   status = pop3_uidl_all (pop3, &iterator);
   if (status == 0)
    {
        for (iterator_first (iterator);
             !iterator_is_done (iterator);
             iterator_next (iterator))
          {
              unsigned int msgno;
              char *uidl;
              if (pop3_uidl_current (iterator, &msgno, &uidl) == 0)
               {
                   printf ("LIST: %d %s\n", msgno, uidl);
                   free (uidl);
               }
          }
        iterator (&iterator);
    }
   else
     printf ("NONE\n");
}
```

MU_ERROR_INVALID_PARAMETER

MU_ERROR_IO

MU_ERROR_TIMEOUT

MU_ERROR_TRY_AGAIN

MU_ERROR_OPERATION_DENIED

int **pop3_uidl_current** (iterator_t *iterator*, unsigned int **msgno*, char ***uidl*) Function

Cover functions around `iterator_current ()` from an iterator created by `pop3_uidl_all ()` to format the result. The variable *uidl* should be `free ()`'ed.

MU_ERROR_INVALID_PARAMETER

MU_ERROR_IO

MU_ERROR_TIMEOUT

MU_ERROR_TRY_AGAIN

MU_ERROR_OPERATION_DENIED

2.1.1.15 User

int **pop3_user** (pop3_t, const char *user) Function

Sends the User command.

MU_ERROR_INVALID_PARAMETER

MU_ERROR_IO

MU_ERROR_TIMEOUT

MU_ERROR_TRY_AGAIN

MU_ERROR_OPERATION_DENIED

2.1.1.16 Help functions

int **pop3_writeline** (pop3_t, const char *format, ...); Function

Copy in the internal buffer of `pop3_t` the string, `pop3_send ()` should be called later to flush the string to the POP3 server.

MU_ERROR_INVALID_PARAMETER

MU_ERROR_IO

MU_ERROR_TIMEOUT

MU_ERROR_TRY_AGAIN

MU_ERROR_OPERATION_DENIED

int **pop3_sendline** (pop3_t, const char *cmd); Function

Cover function for `pop3_writeline ()` and `pop3_send ()`.

MU_ERROR_INVALID_PARAMETER

MU_ERROR_IO

MU_ERROR_TIMEOUT

MU_ERROR_TRY_AGAIN

MU_ERROR_OPERATION_DENIED

int **pop3_send** (pop3_t, const char *cmd); Function

Flushes out the strings written by `pop3_writeline ()` in the internal buffer to the channel.

MU_ERROR_INVALID_PARAMETER

MU_ERROR_IO

MU_ERROR_TIMEOUT

MU_ERROR_TRY_AGAIN

MU_ERROR_OPERATION_DENIED

int **pop3_response** (pop3_t, char *buffer, size_t len, size_t *plen) Function

The last response from the last command is save and can be examine after a failure or success.

MU_ERROR_INVALID_PARAMETER

`MU_ERROR_IO`

`MU_ERROR_TIMEOUT`

`MU_ERROR_TRY_AGAIN`

`MU_ERROR_OPERATION_DENIED`

2.1.1.17 Timeout

`int` **pop3_set_timeout** (`pop3_t, int` *timeout*) Function

Set the timeout time for I/O on the carrier. The default is 10 minutes. The *timeout* is given in milliseconds.

`int` **pop3_get_timeout** (`pop3_t, int *`*timeout*) Function

Get the timeout time for I/O on the carrier. The *timeout* is given in milliseconds.

2.2 IMAP4

```
/* Prefix imap4_ is reserved */
#include <mailutils/imap4.h>
```

Internet Message Access Protocol - Version (4rev1). Not implemented.

2.2.1 Commands

2.2.1.1 Initialization

`int` **imap4_create** (`imap4_t *`) Function

`int` **imap4_open** (`imap4_t, const char *`*hostname*`, unsigned int` *port*`, int` *flags*) Function

`int` **imap4d_set_timeout** (`imap4_t, unsigned int` *seconds*) Function

2.2.1.2 Append

`int` **imap4_append** (`imap4_t`) Function

2.2.1.3 Capability

`int` **imap4_capability** (`imap4_t`) Function

2.2.1.4 Create

`int` **imap4_create_mailbox** (`imap4_t, const char *`*mbox*) Function

2.2.1.5 Check

int **imap4_check** (imap4_t) Function

2.2.1.6 Close

int **imap4_close** (imap4_t) Function

2.2.1.7 Copy

int **imap4_copy** (imap4_t) Function

2.2.1.8 UID Copy

int **imap4_uid_copy** (imap4_t) Function

2.2.1.9 Delete

int **imap4_delete** (imap4_t) Function

2.2.1.10 Fetch

int **imap4_fetch** (imap4_t) Function

2.2.1.11 UID Fetch

int **imap4_uid_fetch** (imap4_t) Function

2.2.1.12 Examine

int **imap4_examine** (imap4_t) Function

2.2.1.13 Expunge

int **imap4_expunge** (imap4_t) Function

2.2.1.14 List

int **imap4_list** (imap4_t) Function

2.2.1.15 Lsub

int **imap4_lsub** (imap4_t) Function

2.2.1.16 Namespace

int **imap4_namespace** (imap4_t) Function

2.2.1.17 Rename

int **imap4_rename** (imap4_t) Function

2.2.1.18 Search

int **imap4_search** (imap4_t) Function

2.2.1.19 UID Search

int **imap4_uid_search** (imap4_t) Function

2.2.1.20 Select

int **imap4_select** (imap4_t) Function

2.2.1.21 Status

int **imap4_status** (imap4_t) Function

2.2.1.22 Store

int **imap4_store** (imap4_t) Function

2.2.1.23 UID Store

int **imap4_uid_store** (imap4_t) Function

2.2.1.24 Subscribe

int **imap4_subscribe** (imap4_t) Function

2.2.1.25 Unsubscribe

int **imap4_unsubscribe** (imap4_t) Function

2.3 Mbox

```
/* Prefix mbox_ is reserved */
#include <mailutils/mbox.h>
```

The most common mailbox format on UNIX platform is *mbox*. Mbox file is messages separated by the special format string.

```
From SP envelope-sender SP date [SP moreinfo]
```

`"From "` is sometimes call the *From_*.

`envelope-sender`
: is a word with no space.

`date` has the same format as `asctime ()`

`moreinfo` are optional values that are seldom used.

A *mbox_t* is created, initialized and destroyed by `mbox_create ()` and `mbox_destroy ()`. When opening, `mbox_open ()` will do a quick check to see if the format is a valid format or an empty file. The scanning of the mailbox is done by `mbox_scan ()`. The function, `mbox_scan ()`, takes callback functions called during the scanning to provide information. The scanning will cache some of the headers fields for speed. Closing the mailbox, `mbox_close ()` will free any resources like, headers cache, locks etc ... All the messages with attributes marked deleted will only be removed on `mbox_expunge ()`. If only the attributes need to be save but the messages not removed, this can be done by `mbox_save_attributes ()`. New messages are added with `mbox_append ()`. Attributes are saved in the *Status:* header field, Read is 'R', Seen is 'O', Deleted is 'd' and Reply is 'r'.

2.3.0.1 Initialization

`int` **mbox_create** (`mbox_t *`*mbox*) Function

Allocate and initialize a *mbox* handle.

`MU_ERROR_NO_MEMORY`

`MU_ERROR_INVALID_PARAMETER`

`void` **mbox_destroy** (`mbox_t` *mbox*) Function

When a POP3 session is finished, the structure must be `free ()`'ed to reclaim memory.

2.3.0.2 Carrier

`int` **mbox_set_carrier** (`mbox_t, stream_t` *carrier*); Function

Another type of stream can be provided to work on, the *carrier* is set in the *mbox_t* handle. Any previous *carrier* stream in the handle, will be close and release. Since the parsing code maintain only the offsets off the message the *carrier* stream must be seekable.

`MU_ERROR_INVALID_PARAMETER`

`int` **mbox_get_carrier** (`mbox_t, stream_t` **carrier*); Function

Return the *mbox_t* carrier. If none was set, a new file stream will be created.

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_NO_MEMORY`

`int` **mbox_open** (`mbox_t, const char` **filename*, `int` *flags*) Function

Open carrier stream with *filename* and *flags*. The stream will be quickly examine to see if it is a mbox format.

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_NO_MEMORY`

`MU_ERROR_NO_ENTRY`

`MU_ERROR_NO_ACCESS`

`MU_ERROR_NOT_SUPPORTED`

`int` **mbox_close** (`mbox_t`) Function

Close the carrier stream and resources particular to the mailbox.

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_NO_MEMORY`

`int` **mbox_uidnext** (`mbox_t, unsigned long` **uidnext*) Function

Return the uidnext, if the *mbox_t* was not scan `mbox_scan ()` is called first.

`MU_ERROR_INVALID_PARAMETER`

`same as mbox_scan ()`

`int` **mbox_uidvalidity** (`mbox_t, unsigned long` **uidvalidity*) Function

Return the uidvalidity, if the *mbox_t* was not scan `mbox_scan ()` is called first.

`MU_ERROR_INVALID_PARAMETER`

`same as mbox_scan ()`

`int` **mbox_get_attribute** (`mbox_t, unsigned int` *msgno*, `attribute_t` **attribute*) Function

Return an *attribute* to indicate the status of message number *msgno*.

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_NO_MEMORY`

`int` **mbox_get_separator** (`mbox_t, unsigned int` *msgno*, `char` ***sep*) Function

Return an allocated string in *sep* containing the value "From " separating each message in Unix mbox format. The string should be `free ()`ed by the caller.

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_NO_MEMORY`

`int` **mbox_set_separator** (`mbox_t, unsigned int` *msgno*, `const char` **sep*) — Function

The variable *sep* should contain a valid "From " separator that will be use when the expunging.

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_NO_MEMORY`

`int` **mbox_get_hstream** (`mbox_t, unsigned int` *msgno*, `stream_t` **stream*) — Function

Return a *stream* to read the header of message *msgno*. The *stream* should be destroy after usage.

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_NO_MEMORY`

`int` **mbox_set_hstream** (`mbox_t, unsigned int` *msgno*, `stream_t` *stream*) — Function

Use *stream* when expunging for message *msgno*.

`MU_ERROR_INVALID_PARAMETER`

`int` **mbox_set_hsize** (`mbox_t, unsigned int` *msgno*, `unsigned int` **size*) — Function

Return the *size* of message *msgno*.

`MU_ERROR_INVALID_PARAMETER`

`int` **mbox_set_hlines** (`mbox_t, unsigned int` *msgno*, `unsigned int` **size*) — Function

Return the number of *lines* of message *msgno*.

`MU_ERROR_INVALID_PARAMETER`

`int` **mbox_get_bstream** (`mbox_t, unsigned int` *msgno*, `stream_t` **stream*) — Function

Return a *stream* to read the body of message *msgno*. The *stream* should be destroy after usage.

`MU_ERROR_INVALID_PARAMETER`

`MU_ERROR_NO_MEMORY`

`int` **mbox_set_bstream** (`mbox_t, unsigned int` *msgno*, `stream_t` *stream*) — Function

Use *stream* when expunging for message *msgno*.

`MU_ERROR_INVALID_PARAMETER`

`int` **mbox_set_bsize** (`mbox_t, unsigned int` *msgno*, `unsigned int` **size*) — Function

Return the *size* of message *msgno*.

`MU_ERROR_INVALID_PARAMETER`

int **mbox_set_blines** (mbox_t, unsigned int *msgno*, unsigned int **size*) Function

Return the number of *lines* of message *msgno*.

`MU_ERROR_INVALID_PARAMETER`

int **mbox_get_size** (mbox_t, unsigned int **size*) Function

Return the *size* of mailbox.

`MU_ERROR_INVALID_PARAMETER`

int **mbox_save** (mbox_t) Function

Save the changes to the messages back to the mailbox, but do not remove messages mark for deletion in the process.

`MU_ERROR_INVALID_PARAMETER`

int **mbox_mak_deleted** (mbox_t, unsigned int *msgno*) Function

Mark *msgno* for deletion.

`MU_ERROR_INVALID_PARAMETER`

int **mbox_unmak_deleted** (mbox_t, unsigned int *msgno*) Function

Unmark *msgno* if it was marked for deletion.

`MU_ERROR_INVALID_PARAMETER`

int **mbox_expunge** (mbox_t) Function

Save the changes to the mailbox and in the process remove all messages marked for deletion.

`MU_ERROR_INVALID_PARAMETER`

int **mbox_append** (mbox_t, const char **sep*, stream_t *stream*) Function

Append to the mailbox an rfc822 message represented by *stream*. The variable *sep* should contain a valid "From " separator or NULL to get the default.

`MU_ERROR_INVALID_PARAMETER`

int **mbox_append_hb** (mbox_t, const char **sep*, stream_t *hstream*, stream_t *bstream*) Function

Append to the mailbox an rfc822 message represented by a header, *hstream*, and a body, *bstream*. The variable *sep* should contain a valid "From " separator or NULL to get the default.

`MU_ERROR_INVALID_PARAMETER`

int **mbox_scan** (mbox_t, unsigned int start, unsigned int **count*) Function

Start scanning the mailbox for new messages. The variable *start* can be a message number starting point. The result of the scanning will be in *count*. The scanning will trigger the `mbox_newmsg_cb()` callback for each new message and `mbox_progress_cb ()` at different interval to notify progression. The return values of the those callback should be 0 is different then 0 the scanning will be stop an the function returns MU_ERROR_INTERRUPTED.

`MU_ERROR_INVALID_PARAMETER`
`MU_ERROR_INTERRUPTED`
`MU_ERROR_NO_MEMORY`

`int` **mbox_set_progress_cb** (`mbox_t`, `int` (***callback*) (`int`, `void *`)), `void` **arg*) — Function

Set the callback function for progress. The variable *arg* will be pass back in the callback as the second argument.

`MU_ERROR_INVALID_PARAMETER`

`int` **mbox_set_newmsg_cb** (`mbox_t`, `int` (***callback*) (`int`, `void *`)), `void` **arg*) — Function

Set the callback function for new messages. The variable *arg* will be pass back in the callback as the second argument.

`MU_ERROR_INVALID_PARAMETER`

2.4 Mh

```
/* Prefix mh_ is reserved */
#include <mailutils/mh.h>
```

Mail Handler mailbox format. Not implemented.

2.5 Maildir

```
/* Prefix maildir_ is reserved */
#include <mailutils/maildir.h>
```

QMail mailbox format. Not implemented.

2.6 SMTP

```
/* Prefix smtp_ is reserved */
#include <mailutils/smtp.h>
```

Simple Mail Transfer Protocol. Not implemented.

2.6.1 Commands

2.6.1.1 Initialization

`int` **smtp_create** (`smtp_t *`) — Function

`void` **smtp_destroy** (`smtp_t *`) — Function

`int` **smtp_open** (`smtp_t`, `const char` **host*, `unsigned int` *port*, `int` *flags*) — Function

2.6.1.2 Data

int **smtp_data** (smtp_t, stream_t *stream*) Function

2.6.1.3 Helo

int **smtp_helo** (smtp_t, const char **domain*) Function

int **smtp_ehlo** (smtp_t, const char **domain*) Function

2.6.1.4 Expn

int **smtp_expn** (smtp_t, const char **list*, iterator_t *) Function

2.6.1.5 Help

int **smtp_help** (smtp_t, const char **help*, iterator_t *) Function

2.6.1.6 Mail From

int **smtp_mail_from** (smtp_t, const char **address*, const char **param*) Function

2.6.1.7 Noop

int **smtp_noop** (smtp_t) Function

2.6.1.8 Quit

int **smtp_quit** (smtp_t) Function

2.6.1.9 Recpt To

int **smtp_rcpt_to** (smtp_t, const char **address*, const char **param*) Function

2.6.1.10 Reset

int **smtp_reset** (smtp_t) Function

2.6.1.11 Verify

int **smtp_verify** (smtp_t, const char **user*) Function

2.6.1.12 Help functions

extern **int** smtp_readline (smtp_t, char **buffer*, size_t *len*, size_t **len*) — Function

extern **int** smtp_response (smtp_t, char **buffer*, size_t *len*, size_t **len*) — Function

extern **int** smtp_writeline (smtp_t, const char **format*, ...) — Function

extern **int** smtp_sendline (smtp_t, const char **line*) — Function

extern **int** smtp_send (smtp_t — Function

2.7 Sendmail

```
/* Prefix sendmail_ is reserved */
#include <mailutils/sendmail.h>
```

Spawning Sendmail daemon to deliver mail. Not implemented.

2.8 NNTP

```
/* Prefix nntp_ is reserved */
#include <mailutils/nntp.h>
```

Network News Transfer Protocol. Not implemented.

2.8.1 Commands

2.8.1.1 Initialization

int **nntp_create** (nnpt_t *) — Function

int **nntp_destroy** (nnpt_t *) — Function

int **nntp_open** (nnpt_t) — Function

2.8.1.2 Article

int **nntp_article** (nnpt_t) — Function

2.8.1.3 Body

int **nntp_body** (nntp_t) — Function

2.8.1.4 Group

int **nntp_group** (nntp_t) Function

2.8.1.5 Head

int **nntp_head** (nntp_t) Function

2.8.1.6 Help

int **nntp_help** (nntp_t) Function

2.8.1.7 IHave

int **nntp_ihave** (nntp_t) Function

2.8.1.8 Last

int **nntp_last** (nntp_t) Function

2.8.1.9 List

int **nntp_list** (nntp_t) Function

2.8.1.10 NewGroups

int **nntp_newgroups** (nntp_t) Function

2.8.1.11 NewNews

int **nntp_newnews** (nntp_t) Function

2.8.1.12 Next

int **nntp_next** (nntp_t) Function

2.8.1.13 Post

int **nntp_post** (nntp_t) Function

2.8.1.14 Quit

int **nntp_quit** (nntp_t) Function

2.8.1.15 Slave

int **nntp_slave** (nntp_t) Function

2.8.1.16 Stat

int **nntp_stat** (nntp_t) Function

2.9 Parse822

```
/* Prefix parse822_ is reserved */
#include <mailutils/parse822.h>
```

Internet Message Format, see Address node for the discussion.

int **parse822_address_list** (address_t* a, const char* s) Function

int **parse822_mail_box** (const char** p, const char* e, address_t* a) Function

int **parse822_group** (const char** p, const char* e, address_t* a) Function

int **parse822_address** (const char** p, const char* e, address_t* a) Function

int **parse822_route_addr** (const char** p, const char* e, address_t* a) Function

int **parse822_route** (const char** p, const char* e, char** route) Function

int **parse822_addr_spec** (const char** p, const char* e, address_t* a) Function

int **parse822_unix_mbox** (const char** p, const char* e, address_t* a) Function

int **parse822_local_part** (const char** p, const char* e, char** local_part) Function

int **parse822_domain** (const char** p, const char* e, char** domain) Function

int **parse822_sub_domain** (const char** p, const char* e, char** sub_domain) Function int **parse822_domain_ref** (const char** p, const char* e, char** domain_ref) Function

int **parse822_domain_literal** (const char** p, const char* e, char** domain_literal) Function

int **parse822_quote_string** (char** quoted, const char* raw) Function

int **parse822_quote_local_part** (char** quoted, const char* raw) Function

int **parse822_field_body** (const char** p, const char *e, char** fieldbody) Function

int **parse822_field_name** (const char** p, const char *e, char** fieldname) Function

3 Framework

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

```
                         folder_t                  url_t
 -/var/mail-    +- .. ->+-----------------+   +-->+------------+
(  alain *-)-+  |       |   url_t       *-|---+   |   port     |
 ----------- |  |       |-----------------|       |   hostname |
(  jakob *-)-+--+       |   auth_t      *-|---+   |   file     |
 ----------- |          |-----------------|   |   |   ...      |
(  jeff  *-)-+          |   stream_t      |   |   +------------+
 ----------- |          |-----------------|   |
(  shaleh*-)-+          |   .....         |   |    auth_t
 ----------             |-----------------|   +-->+------------+
                    +---|-* mailbox_t[]   |       |   ticket_t |
 mailbox_t          |   +-----------------+       +------------+
+----------------+<-+
|   locker_t  *--|-------------+
|----------------|             |
|   url_t        |             |          locker_t
|----------------|             +--------->+---------+
|   stream_t     |                        | lock    |
|----------------|                        | unlock  |
|  message_t[] *-|-------+                +---------+
+----------------+       |                     envelope_t
                         |          +--------->+-----------+
 message_t               |          |         | date      |
+----------------+<------+          |         | from      |
|   envelope_t *-|------------------+         | to        |
|----------------|              header_t      +-----------+
|   header_t   *-|------------>+--------------+
|----------------|             |   stream_t   |
|   body_t     *-|----+        +--------------+
+----------------+    |    body_t
                      +-->+--------------+
                          |   stream_t   |
                          +--------------+
```

For example writing a simple `from` command that will list the *From* and *Subject* headers of every mail in a mailbox.

```
/* sfrom, Simple From */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <mailutils/registrar.h>
#include <mailutils/list.h>
#include <mailutils/mailbox.h>

#define BUFFER_SIZE 64
int
main (int argc, const char **argv)
{
```

```
char from[BUFFER_SIZE];
char subject[BUFFER_SIZE];
char *mail;
mailbox_t mbox;
int status
size_t msgno, total = 0;

mail = (argc == 2) ? argv[1] : getenv ("MAIL");

/* Register the type of mailbox. IMAP4, POP3 and local format  */
{
    list_t registrar;
    registrar_get_list (&registrar);
    list_append (registrar, imap_record);
    list_append (registrar, path_record);
    list_append (registrar, pop_record);
}

status = mailbox_create (&mbox, mail);
if (status != 0)
  {
     fprintf (stderr, "mailbox_create: %s\n", strerror (status));
     exit (EXIT_FAILURE);
  }

status = mailbox_open (mbox, MU_STREAM_READ);
if (status != 0)
  {
     fprintf (stderr, "mailbox_open: %s\n", strerror (status));
     exit (EXIT_FAILURE);
  }

mailbox_messages_count (mbox, &total);

for (msgno = 1; msgno <= total; msgno++)
   {
      message_t msg;
      header_t hdr;

      if ((status = mailbox_get_message (mbox, msgno, &msg)) != 0
          || (status = message_get_header (msg, &hdr)) != 0)
        {
            fprintf (stderr, "Error message:%s\n",
                     strerror (status));
            exit (EXIT_FAILURE);
        }

      status = header_get_value (hdr, MU_HEADER_FROM, from,
                                 sizeof (from), NULL);
```

```
        if (status != 0)
           strcpy (from, "(NO FROM)");

        status = header_get_value (hdr, MU_HEADER_SUBJECT, subject,
                                   sizeof (subject), NULL);
        if (status != 0)
           strcpy (subject, "(NO SUBJECT)");

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

    mailbox_close (mbox);
    mailbox_destroy (&mbox);
    return 0;
}
```

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

3.1 Folder

```
/* Prefix folder_ is reserve */
#include <mailutils/folder.h>
```

```
                               folder_t                  url_t
  -/var/mail-    +---//---->/-----------------\    +-->/-----------\
 (  alain *-)-+  |          |  url_t       *-|----+    |  port      |
  ----------- |  |          |-----------------+        |  hostname  |
 (  jakob *-)-+--+          |  observer_t   *-|        |  file      |
  ----------- |             |-----------------+        |  ...       |
 (  jeff  *-)-+             |  stream_t        |       \-----------/
  ----------- |             |-----------------|
 (  sean  *-)-+             |  auth_t          |
  ----------                |-----------------|
                            |  mailbox_t(1)    |
                            |-----------------|
                            |  mailbox_t(2)    |
                            |  ......          |
                            |  mailbox_t(n)    |
                            \-----------------/
```

int **folder_create** (folder_t *, const char **url*) Function

void **folder_destroy** (folder_t *) Function int **folder_open** (folder_t, int *flag*) Function

int **folder_close** (folder_t) Function

int **folder_delete** (folder_t, const char **mailbox*) Function

int **folder_rename** (folder_t, const char *, const char **mailbox*) Function

int **folder_subscribe** (folder_t, const char **mailbox*) Function

int **folder_unsubscribe** (folder_t, const char **mailbox*) Function

int **folder_list** (folder_t, const char **ref*, const char **wcard*, iterator_t *) Function

int **folder_lsub** (folder_t, const char **ref*, const char **wcar*, iterator_t *) Function

int **folder_get_stream** (folder_t, stream_t *) Function

int **folder_set_stream** (folder_t, stream_t) Function

int **folder_get_observable** (folder_t, observable_t *) Function

int **folder_get_debug** (folder_t, debug_t *) Function

int **folder_set_debug** (folder_t, debug_t) Function

int **folder_get_authority** (folder_t, authority_t *) Function

int **folder_set_authority** (folder_t, authority_t) Function

int **folder_get_url** (folder_t, url_t *) Function

int **folder_set_url** (folder_t, url_t) Function

3.2 Mailbox

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

mailbox_t Data Type

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

```
                              mailbox_t                   url_t
 -/var/mail-    +---//---->/-----------------\    +-->/-----------\
(  alain   )    |          |  url_t       *-|----+   |  port      |
 -----------    |          |-----------------+       |  hostname  |
(  jakob *-)----+          |  observer_t   *-|       |  file      |
 -----------               |-----------------+       |  ...       |
(  jeff    )               |  stream_t       |       \-----------/
 -----------               |-----------------|
(  sean    )               |  locker_t       |
 ----------                |-----------------|
                           |  message_t(1)   |
                           |-----------------|
                           |  message_t(2)   |
                           |  ......         |
                           |  message_t(n)   |
                           \-----------------/
```

`int` **mailbox_append_message** (`mailbox_t` *mbox*, `message_t` *message*) — Function

The *message* is appended to the mailbox *mbox*.

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

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null or *message* is invalid.

`int` **mailbox_close** (`mailbox_t` *mbox*) — Function

The stream attach to *mbox* is closed.

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

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null.

`int` **mailbox_create** (`mailbox_t *`*pmbox*, `const char *`*name*) — Function

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

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

`MU_ERROR_INVALID_PARAMETER`
: The url *name* supplied is invalid or not supported. *pmbox* is NULL.

`ENOMEM`
: Not enough memory to allocate resources.

`int` **mailbox_create_default** (`mailbox_t *`*pmbox*, `const char *`*name*) — Function

Create a mailbox with `mailbox_create ()` based on the environment variable *£MAIL* or the string formed by *_PATH_MAILDIR/user*" or *£LOGNAME* if *user* is null,

`void` **mailbox_destroy** (`mailbox_t *`*pmbox*) — Function

Destroys and releases resources held by *pmbox*.

int **mailbox_expunge** (`mailbox_t` *mbox*) Function

All messages marked for deletion are removed.

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

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null.

int **mailbox_get_folder** (`mailbox_t` *mbox*, `folder_t *`*folder*) Function

Get the *folder*.

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

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null.

int **mailbox_get_debug** (`mailbox_t` *mbox*, `debug_t *`*debug*) Function

Get a debug object. The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null.

`ENOMEM`

int **mailbox_get_locker** (`mailbox_t` *mbox*, `locker_t *`*plocker*) Function

Return the *locker* object.

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

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null.

int **mailbox_get_message** (`mailbox_t` *mbox*, `size_t` *msgno*, `message_t *`*pmessage*) Function

Retrieve message number *msgno*, *pmessage* is allocated and initialized.

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

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null or *msgno* is invalid.

`ENOMEM` Not enough memory.

int **mailbox_get_observable** (`mailbox_t mbox` *mbox*, `observable_t*`*observable*) Function

Get the observable object.

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

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null.

`ENOMEM` Not enough memory.

int **mailbox_get_property** (`mailbox_t` *mbox*, `property_t` **property*) — Function

Get the property object. The return value is `0` on success and a code number on error conditions:

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null.

`ENOMEM`

int **mailbox_get_size** (`mailbox_t` *mbox*, `off_t` **psize*) — Function

Gives the *mbox* size.

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

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null.

int **mailbox_get_stream** (`mailbox_t` *mbox*, `stream_t` **pstream*) — Function

The mailbox stream is put in *pstream*.

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

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is invalid or *pstream* is NULL.

int **mailbox_get_ticket** (`mailbox_t` *mbox*, `ticket_t` *ticket*) — Function

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

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null.

int **mailbox_get_url** (`mailbox_t` *mbox*, `url_t` **purl*) — Function

Gives the constructed *url*.

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

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null.

int **mailbox_is_modified** (`mailbox_t` *mbox*) — Function

Check if the mailbox been modified by an external source.

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

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null.

int **mailbox_message_unseen** (`mailbox_t` *mbox*, `size_t` **pnumber*); — Function

Give the number of first unseen message in *mbox*.

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

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null.

int **mailbox_messages_count** (mailbox_t *mbox*, size_t *pnumber*); — Function

Give the number of messages in *mbox*.

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

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null.

int **mailbox_messages_recent** (mailbox_t *mbox*, size_t *pnumber*); — Function

Give the number of recent messages in *mbox*.

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

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null.

int **mailbox_open** (mailbox_t *mbox*, int *flag*) — Function

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

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

`EAGAIN`
`EINPROGRESS`
: Operation in progress.

`EBUSY`
: Resource busy.

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null or flag is invalid.

`ENOMEM`
: Not enough memory.

int **mailbox_scan** (mailbox_t *mbox*, size_t *msgno*, size_t *pcount*); — Function

Scan the mailbox for new messages starting at message *msgno*.

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

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null.

`ENOMEM`
: Not enough memory.

int **mailbox_set_locker** (mailbox_t *mbox*, locker_t *locker*) — Function

Set the type of locking done by the *mbox*.

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

`MU_ERROR_INVALID_PARAMETER`
: *mbox* is null.

int **mailbox_set_stream** (mailbox_t *mbox*, stream_t *stream*) Function

Set the *stream* connection to use for the mailbox.

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

MU_ERROR_INVALID_PARAMETER
: *mbox* or *stream* is NULL.

int **mailbox_set_ticket** (mailbox_t *mbox*, ticket_t *ticket*) Function

The *ticket* will be set on the auth_t object on creation.

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

MU_ERROR_INVALID_PARAMETER
: *mbox* is null.

int **mailbox_uidnext** (mailbox_t *mbox*, size_t **pnumber*); Function

Give the next predicted uid for *mbox*.

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

MU_ERROR_INVALID_PARAMETER
: *mbox* is null.

int **mailbox_uidvalidity** (mailbox_t *mbox*, size_t **pnumber*); Function

Give the uid validity of *mbox*.

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

MU_ERROR_INVALID_PARAMETER
: *mbox* is null.

3.3 Mailer

```
/* Prefix mailer_ is reserved */
#include <mailutils/mailer.h>
```

The API is still shaky and undefined.

int **mailer_create** (mailer_t *, const char *) Function

void **mailer_destroy** (mailer_t *) Function

int **mailer_open** (mailer_t, int flags) Function

int **mailer_close** (mailer_t) Function

int **mailer_send_message** (mailer_t, message_t) Function

int **mailer_get_property** (mailer_t, property_t *) Function

int **mailer_get_stream** (mailer_t, stream_t *) Function int **mailer_set_stream** (mailer_t, stream_t) Function

int **mailer_get_debug** (mailer_t, debug_t *) Function

int **mailer_set_debug** (mailer_t, debug_t) Function

int **mailer_get_observable** (mailer_t, observable_t *) Function

int **mailer_get_url** (mailer_t, url_t *) Function

3.4 Message

```
#include <mailutils/message.h>
/* Prefix message_ is reserve */
```

The `message_t` object is a convenient way to manipulate messages. It encapsulates the `envelope_t`, the `header_t` and the `body_t`.

```
  mailbox_t
                              message_t
  ----------
 (message[1])        +------>+-----------------------+
  ----------         |       |  envelope_t           |
 (message[2])        |       |-----------------------|
  ----------         |       |  header_t             |
 (message[3])--------+       |-----------------------|
  ----------                 |  body_t               |
 (message[n])                |-----------------------|
  ----------                 |  attribute_t          |
                             |-----------------------|
                             |  stream_t             |
                             +-----------------------+
```

void **message_create** (message_t **msg*, void **owner*) Function

void **message_destroy** (message_t **msg*, void **owner*) Function

The resources allocate for *msg* are freed.

int **message_get_header** (message_t *msg*, header_t **pheader*) Function

Retrieve *msg* header.

int **message_set_header** (message_t *msg*, header_t *header*, void **owner*) Function

int **message_get_body** (message_t *msg*, body_t **pbody*) Function

int **message_set_body** (message_t *msg*, body_t *body*, void **owner*) Function

int **message_is_multipart** (message_t *msg*) Function

Return non-zero value if message is multi-part.

`int` **message_get_num_parts** (`message_t` *msg*, `size_t *nparts`) — Function

`int` **message_get_part** (`message_t` *msg*, `size_t part`, `message_t *msg`) — Function

`int` **message_get_stream** (`message_t` *msg*, `stream_t *`*pstream*) — Function

`int` **message_set_stream** (`message_t` *msg*, `stream_t` *stream*,`void *`*owner*) — Function

`int` **message_get_attribute** (`message_t` *msg*, `attribute_t *`*pattribute*) — Function

`int` **message_set_attribute** (`message_t` *msg*, `attribute_t` *attribute*, `void *owner`) — Function

`int` **message_get_envelope** (`message_t` *msg*, `envelope_t *penvelope`) — Function

`int` **message_set_envelope** (`message_t` *msg*, `envelope_t envelope`, `void *`*owner*) — Function

`int` **message_get_uid** (`message_t` *msg*, `size_t *`*uid*) — Function

`int` **message_get_uidl** (`message_t` *msg*, `char *`*buffer*, `size_t` *buflen*, `size_t *`*pwriten*) — Function

`int` **message_set_uidl** (`message_t` *msg*, `int (*`*_get_uidl*`)(message_t, char *, size_t, size_t *)`, `void *`*owner*) — Function

`int` **message_get_observable** (`message_t` *msg*, `observable_t *`*observable*) — Function

`int` **message_create_attachment** (`const char *`*content_type*, `const char *`*encoding*, `const char *`*filename*, `message_t *`*newmsg*) — Function

`int` **message_save_attachment** (`message_t` *msg*, `const char *`*filename*, `void **`*data*) — Function

`int` **message_encapsulate** (`message_t` *msg*, `message_t *`*newmsg*, `void **`*data*) — Function

`int` **message_unencapsulate** (`message_t` *msg*, `message_t *`*newmsg*, `void **`*data*`);` — Function

3.5 Envelope

```
/* Prefix envelope_ is reserved */
#include <mailutils/envelope.h>
```

`int` **envelope_date** `(envelope_t, char *, size_t, size_t *);` Function

Get the date that the message was delivered to the mailbox, in something close to ANSI ctime() format: Mon Jul 05 13:08:27 1999.

`int` **envelope_sender** `(envelope_t, char *, size_t, size_t *);` Function

Get the address that this message was reportedly received from. This would be the "mail from" argument if the message was delivered or received via SMTP, for example.

`int` **envelope_get_message** `(envelope_t, message_t *);` Function

`int` **envelope_create** `(envelope_t *, void *);` Function

Primarily for internal use.

`void` **envelope_destroy** `(envelope_t *, void *);` Function

Primarily for internal use.

`int` **envelope_set_sender** `(envelope_t, int (*_sender) __P ((envelope_t, char *, size_t, size_t*)), void *);` Function

Primarily for internal use. The implementation of envelope_t depends on the mailbox type, this allows the function which actually gets the sender to be set by the creator of an envelope_t.

`int` **envelope_set_date** `(envelope_t, int (*_date) __P ((envelope_t, char *, size_t, size_t *)), void *);` Function

Primarily for internal use. The implementation of envelope_t depends on the mailbox type, this allows the function which actually gets the date to be set by the creator of an envelope_t.

3.6 Headers

```
/* Prefix header_ is reserved */
#include <mailutils/header.h>
```

So far we plan support for RFC822 and plan for RFC1522. with RFC1522 non ASCII characters will be encoded.

`int` **header_create** `(header_t *`*hdr*`, const char *`*blurb*`, size_t` *len*`, void *`*owner*`)` Function

Initialize a *hdr* to a supported type. If *blurb* is not NULL, it is parsed.

`void` **header_destroy** `(header_t *`*hdr*`, void *`*owner*`)` Function

The resources allocated for *hdr* are freed.

`int` **header_set_value** (`header_t` *hdr*, `const char *`*fn*, `const char *`*fv*, `size_t n`, `int` *replace*) — Function

Set the field-name *fn* to field-value *fv* of size *n* in *hdr*. If *replace* is non-zero the initial value is replaced, if zero it is appended.

Some basic macros are already provided for rfc822.

`MU_HDR_RETURN_PATH`
: Return-Path

`MU_HDR_RECEIVED`
: Received

`MU_HDR_DATE`
: Date

`MU_HDR_FROM`
: From

`MU_HDR_RESENT_FROM`
: Resent-From

`MU_HDR_SUBJECT`
: Subject

`MU_HDR_SENDER`
: Sender

`MU_HDR_RESENT_SENDER`
: Resent-SENDER

`MU_HDR_TO`
: To

`MU_HDR_RESENT_TO`
: Resent-To

`MU_HDR_CC`
: Cc

`MU_HDR_RESENT_CC`
: Resent-Cc

`MU_HDR_BCC`
: Bcc

`MU_HDR_RESENT_BCC`
: Resent-Bcc

`MU_HDR_REPLY_TO`
: Reply-To

`MU_HDR_RESENT_REPLY_TO`
: Resent-Reply-To

`MU_HDR_MESSAGE_ID`
: Message-ID

`MU_HDR_RESENT_MESSAGE_ID`
Resent-Message-ID

`MU_HDR_IN_REPLY_TO`
In-Reply-To

`MU_HDR_ENCRYPTED`
Encrypted

`MU_HDR_PRECEDENCE`
Precedence

`MU_HDR_STATUS`
Status

`MU_HDR_CONTENT_LENGTH`
Content-Length

`MU_HDR_CONTENT_TYPE`
Content-Type

`MU_HDR_MIME_VERSION`
MIME-Version

`int` **header_get_value** (`header_t` *hdr*, `const char *`*fn*, `char *`*fv*, `size_t` *len*, `size_t *`*n*) — Function

Value of field-name *fn* is returned in buffer *fv* of size *len*. The number of bytes written is put in *n*.

`int` **header_aget_value** (`header_t` *hdr*, `const char *`*fn*, `char **`*fv*) — Function

The value is allocated.

`int` **header_get_stream** (`header_t` *hdr*, `stream_t *`*pstream*) — Function

`int` **header_set_size** (`header_t` *hdr*, `size_t *`*size*) — Function

`int` **header_set_lines** (`header_t` *hdr*, `size_t *`*lpines*) — Function

3.7 Body

```
/* Prefix body_ is reserved */
#include <mailutils/body.h>
```

`int` **body_create** (`body_t *`*body*, `void *`*owner*) — Function

Initialize an object *bdy*.

`void` **body_destroy** (`body_t *`*pbody*) — Function

The resources allocate are release.

`int` **body_get_stream** (`body_t` *body*, `stream_t *`*pstream*) — Function int **body_set_stream** (body_t *body*, stream_t *stream*, void **owner*) Function

int **body_get_filename** __P ((body_t *body*, char **buffer*, size_t*buflen*, size_t **pwriten*) Function

int **body_set_filename** (body_t *body*, const char**buffer*) Function

int **body_size** (body_t *body*, size_t**psize*) Function

int **body_lines** (body_t *body*, size_t **plines*) Function

3.8 Attribute

```
/* Prefix attribute_ is reserved */
#include <mailutils/attribute.h>
```

int **attribute_create** (attribute_t **pattribute*) Function

void **attribute_destroy** (attribute_t **pattribute*) Function

int **attribute_is_seen** (attribute_t *attribute*) Function

int **attribute_is_answered** (attribute_t *attribute*) Function

int **attribute_is_flagged** (attribute_t *attribute*) Function

int **attribute_is_deleted** (attribute_t *attribute*) Function

int **attribute_is_draft** (attribute_t *attribute*) Function

int **attribute_is_recent** (attribute_t *attribute*) Function

int **attribute_is_read** (attribute_t *attribute*) Function

int **attribute_set_seen** (attribute_t *attribute*) Function

int **attribute_set_answered** (attribute_t *attribute*) Function

int **attribute_set_flagged** (attribute_t *attribute*) Function

int **attribute_set_deleted** (attribute_t *attribute*) Function

int **attribute_set_draft** (attribute_t *attribute*) Function

int **attribute_set_recent** (attribute_t *attribute*) Function

int **attribute_set_read** (attribute_t *attribute*) Function int **attribute_unset_seen** (attribute_t *attribute*) Function

int **attribute_unset_answered** (attribute_t *attribute*) Function

int **attribute_unset_flagged** (attribute_t *attribute*) Function

int **attribute_unset_deleted** (attribute_t *attribute*) Function

int **attribute_unset_draft** (attribute_t *attribute*) Function

int **attribute_unset_recent** (attribute_t *attribute*) Function

int **attribute_unset_read** (attribute_t *attribute*) Function

int **attribute_is_equal** (attribute_t *att1*, attribute_t *att2*) Function

int **attribute_copy** (attribute_t *dst*, attribute_t *src*) Function

int **string_to_attribute** (const char **buf*, attribute_t **pattr*) Function

int **attribute_to_string** (attribute_t *attr*, char **buf*, size_t *len*, size_t **pwriten*) Function

3.9 Stream

```
#include <mailutils/stream.h>
```

int **stream_create** (stream_t **pstream*, int *flags*, void **owner*) Function

MU_STREAM_READ
: The stream is open read only.

MU_STREAM_WRITE
: The stream is open write only.

MU_STREAM_RDWR
: The stream is open read and write.

MU_STREAM_APPEND
: The stream is open in append mode for writing.

MU_STREAM_CREAT
: The stream is created.

MU_STREAM_NONBLOCK
: The stream is set non blocking.

MU_STREAM_NO_CHECK
: Stream is destroyed without checking for the owner.

void **stream_destroy** (stream_t **pstream*, void **owner*) Function int **stream_open** (stream_t *stream*, const char **name*, int*port*, int *flag*) Function

int **stream_close** (stream_t *stream*) Function

int **stream_get_fd** (stream_t *stream*, int **pfd*) Function

int **stream_read** (stream_t *stream*, char **buffer*, size_t *buflen*, off_t *offset*, size_t **pwriten*) Function

int **stream_readline** (stream_t *stream*, char **buffer*, size_t *buflen*, off_t *offset*, size_t **pwriten*) Function

int **stream_size** (stream_t *stream*, off_t **psize*) Function

int **stream_truncate** (stream_t *stream*, off_t *size*) Function

int **stream_write** (stream_t *stream*, const char **buffer*, size_t *buflen*, off_t *offset*, size_t **pwriten*) Function

int **stream_flush** (stream_t *stream*) Function

int **stream_get_flags** (stream_t *stream*, int **pflags*) Function

int **stream_get_state** (stream_t *stream*, int **pstate*) Function

`MU_STREAM_STATE_OPEN`
: Last action was `stream_open`.

`MU_STREAM_STATE_READ`
: Last action was `stream_read` or `stream_readline`.

`MU_STREAM_STATE_WRITE`
: Last action was `stream_write`.

`MU_STREAM_STATE_CLOSE`
: Last action was `stream_close`.

int **file_stream_create** (stream_t **pstream*) Function

int **mapfile_stream_create** (stream_t **pstream*) Function

int **encoder_stream_create** (stream_t **pstream*, stream_t *iostream*, const char **encoding*) Function

int **decoder_stream_create** (stream_t **pstream*, stream_t *iostream*, const char **encoding*) Function

int **tcp_stream_create** (stream_t **pstream*) Function

An example using `tcp_stream_create` to make a simple web client:

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <sys/select.h>

#include <mailutils/io.h>

const char *wbuf = "GET / HTTP/1.0\r\n\r\n";
char rbuf[1024];

int
main(int argc, char **argv)
{
  int ret, off = 0, fd;
  stream_t stream;
  size_t nb;
  fd_set fds;

  argc = argc, argv = argv;

  ret = tcp_stream_create (&stream);
  if (ret != 0)
    {
      fprintf (stderr, "tcp_stream_create: %s\n",
               mailutils_error(ret));
      exit (EXIT_FAILURE);
    }

connect_again:
  ret = stream_open (stream, "www.netscape.com", 80,
                     MU_STREAM_NONBLOCK);
  if (ret != 0)
    {
      if (ret == MU_ERROR_EAGAIN)
        {
          ret = stream_get_fd(stream, &fd);
          if (ret != 0)
            {
              fprintf (stderr, "stream_get_fd: %s\n",
                       mailutils_error(ret));
              exit (EXIT_FAILURE);
            }
            FD_ZERO (&fds);
            FD_SET (fd, &fds);
            select (fd+1, NULL, &fds, NULL, NULL);
            goto connect_again;
```

```
          }
          fprintf (stderr, "stream_open: %s\n", mailutils_error (ret));
          exit (EXIT_FAILURE);
    }

    ret = stream_get_fd (stream, &fd);
    if (ret != 0)
      {
         fprintf(stderr, "stream_get_fd: %s\n", strerror(ret));
         exit (EXIT_FAILURE);
      }

write_again:
    ret = stream_write (stream, wbuf + off, strlen (wbuf), 0, &nb);
    if (ret != 0 )
      {
         if (ret == EAGAIN)
           {
              FD_ZERO (&fds);
              FD_SET (fd, &fds);
              select (fd + 1, NULL, &fds, NULL, NULL);
              off += nb;
              goto write_again;
           }
         fprintf (stderr, "stream_write: %s\n", strerror(ret));
         exit (EXIT_FAILURE)
      }

      if (nb != strlen (wbuf))
        {
           fprintf(stderr, "stream_write: %s\n", "nb != wbuf length");
           exit (EXIT_FAILURE);
        }

      do
        {
read_again:
           ret = stream_read (stream, rbuf, sizeof (rbuf), 0, &nb);
           if (ret != 0)
             {
                if (ret == EAGAIN)
                  {
                     FD_ZERO (&fds);
                     FD_SET (fd, &fds);
                     select (fd + 1, &fds, NULL, NULL, NULL);
                     goto read_again;
                   }
                 fprintf (stderr, "stream_read: %s\n", strerror(ret));
                 exit(EXIT_FAILURE);
```

```
                }
                write (2, rbuf, nb);
            } while (nb);

            ret = stream_close (stream);
            if (ret!= 0)
              {
                fprintf (stderr, "stream_close: %s\n", strerror(ret));
                exit (EXIT_FAILURE);
              }

            stream_destroy (&stream, NULL);
            exit (EXIT_SUCCESS);
}
```

3.10 Iterator

```
/* Prefix iterator_ is reserved */
#include <mailutils/iterator.h>
```

int **iterator_create** (iterator_t *) Function

void **iterator_destroy** (iterator_t *) Function

int **iterator_first** (iterator_t) Function

int **iterator_next** (iterator_t) Function

int **iterator_current** (iterator_t, void **pitem) Function

int **iterator_is_done** (iterator_t) Function

3.11 Authenticator

```
/* Prefix auth_ is reserved */
#include <mailutils/auth.h>
```

There are many ways to authenticate to a server. To be flexible the authentication process is provided by two objects `auth_t` and `ticket_t`. The `auth_t` can implement different protocol like APOP, MD5-AUTH, One Time Passwd etc .. By default if a mailbox does not understand or know how to authenticate it falls back to user/passwd authentication. The `ticket_t` is a way for Mailboxes and Mailers provide a way to authenticate when the URL does not contain enough information. The default action is to call the function `auth_authenticate` which will get the *user* and *passwd* if not set, this function can be overridden by a custom method.

int **auth_create** (auth_t **pauth*, void **owner*) Function void **auth_destroy** (auth_t **pauth*, void **owner*) Function

int **auth_prologue** (auth_t *auth*) Function

int **auth_authenticate** (auth_t *auth*, char ***user*, char ***passwd*) Function

int **auth_epilogue** (auth_t *auth*) Function

A simple example of an authenticate function:

```
#include <mailutils/auth.h>
#include <stdio.h>
#include <string.h>

int
my_authenticate (auth_t auth, char **user, char **passwd)
{
  char u[128] = "";
  char p[128] = "";

  /* prompt the user name */
  printf ("User: ");
  fflush (stdout);
  fgets (u, sizeof (u), stdin);
  u[strlen (u) - 1] = '\0'; /* nuke the trailing NL */

  /* prompt the passwd */
  printf ("Passwd: "); fflush (stdout);
  echo_off ();
  fgets (p, sizeof(p), stdin);
  echo_on ();
  p[strlen (p) - 1] = '\0';

  /* duplicate */
  *user = strdup (u);
  *passwd = strdup (p);
  return 0;
}
```

3.12 Address

```
/* Prefix address_ is reserved */
#include <mailutils/address.h>
```

The Internet address format is defined in RFC 822. RFC 822 has been updated, and is now superceeded by RFC 2822, which makes some corrections and clarifications. References to RFC 822 here apply equally to RFC 2822.

The RFC 822 format is more flexible than many people realize, here is a quick summary of the syntax this parser implements, see RFC 822 for the details. "[]" pairs mean "optional", "/" means "one or the other", and double-quoted characters are literals.

```
addr-spec    = local-part "¨ domain
mailbox      = addr-spec ["(" display-name ")"] /
               [display-name] "<" [route] addr-spec ">"
mailbox-list = mailbox ["," mailbox-list]
group        = display-name ":" [mailbox-list] ";"
address      = mailbox / group / unix-mbox
address-list = address ["," address-list]
```

unix-mbox is a non-standard extension meant to deal with the common practice of using user names as addresses in mail utilities. It allows addresses such as "root" to be parsed correctly. These are NOT valid internet email addresses, they must be qualified before use.

Several address functions have a set of common arguments with consistent semantics, these are described here to avoid repetition.

Since an address-list may contain multiple addresses, they are accessed by a **one-based** index number, *no*. The index is one-based because pop, imap, and other message stores commonly use one-based counts to access messages and attributes of messages.

If *len* is greater than `0` it is the length of the buffer *buf*, and as much of the component as possible will be copied into the buffer. The buffer will be null terminated.

The size of a particular component may be queried by providing `0` for the *len* of the buffer, in which case the buffer is optional. In this case, if *n* is provided **n* is assigned the length of the component string.

address_t — Data Type

The `address_t` object is used to hold information about a parsed RFC822 address list, and is an opaque data structure to the user. Functions are provided to retrieve information about an address in the address list.

`int` **address_create** (`address_t *`*addr*, `const char *`*string*) — Function

This function allocates and initializes *addr* by parsing the RFC822 address-list *string*.

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

`EINVAL` Invalid usage, usually a required argument was `null`.

`ENOMEM` Not enough memory to allocate resources.

`ENOENT` Invalid RFC822 syntax, parsing failed.

`void` **address_destroy** (`address_t *`*addr*) — Function

The *addr* is destroyed.

`int` **address_get_email** (`address_t *`*addr*, `size_t` *no*, `char*` *buf*, `size_t` *len*, `size_t*` *n*) — Function

Accesses the *no*th email address component of the address list. This address is the plain email address, correctly quoted, suitable for using in an smtp dialog, for example, or as the address part of a contact book entry.

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

`EINVAL` Invalid usage, usually a required argument was `null`.

`ENOENT` The index *no* is outside of the range of available addresses.

int **address_get_personal** (address_t **addr*, size_t *no*, char* *buf*, size_t *len*, size_t* *n*) Function

Accesses the display-name describing the *no*th email address. This display-name is optional, so may not be present. If it is not present, but there is an RFC822 comment after the address, that comment will be returned as the personal phrase, as this is a common usage of the comment even though it is not defined in the internet mail standard.

A group is a kind of a special case. It has a display-name, followed by an optional mailbox-list. The display-name will be allocated an address all it's own, but all the other elements (local-part, domain, etc.) will be zero-length. So "a group: ;" is valid, will have a count of 1, but address_get_email(), and all the rest, will return zero-length output.

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

`EINVAL` Invalid usage, usually a required argument was `null`.

`ENOENT` The index *no* is outside of the range of available addresses.

int **address_get_comments** (address_t **addr*, size_t *no*, char* *buf*, size_t *len*, size_t* *n*) Function

Accesses the comments extracted while parsing the *no*th email address. These comments have no defined meaning, and are not currently collected.

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

`EINVAL` Invalid usage, usually a required argument was `null`.

`ENOENT` The index *no* is outside of the range of available addresses.

int **address_get_email** (address_t **addr*, size_t *no*, char* *buf*, size_t *len*, size_t* *n*) Function

Accesses the email addr-spec extracted while parsing the *no*th email address. This will be `0` length for a unix-mbox.

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

`EINVAL` Invalid usage, usually a required argument was `null`.

`ENOENT` The index *no* is outside of the range of available addresses.

int **address_get_local_part** (address_t **addr*, size_t *no*, char* *buf*, size_t *len*, size_t* *n*) Function

Accesses the local-part of an email addr-spec extracted while parsing the *no*th email address.

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

`EINVAL` Invalid usage, usually a required argument was `null`.

`ENOENT` The index *no* is outside of the range of available addresses.

int **address_get_domain** (address_t **addr*, size_t *no*, char* *buf*, size_t *len*, size_t* *n*) Function

Accesses the domain of an email addr-spec extracted while parsing the *no*th email address. This will be 0 length for a unix-mbox.

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

EINVAL Invalid usage, usually a required argument was null.

ENOENT The index *no* is outside of the range of available addresses.

int **address_get_route** (address_t **addr*, size_t *no*, char* *buf*, size_t *len*, size_t* *n*) Function

Accesses the route of an email addr-spec extracted while parsing the *no*th email address. This is a rarely used RFC822 address syntax, but is legal in SMTP as well. The entire route is returned as a string, those wishing to parse it should look at <mailutils/parse822.h>.

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

EINVAL Invalid usage, usually a required argument was null.

ENOENT The index *no* is outside of the range of available addresses.

int **address_is_group** (address_t **addr*, size_t *no*, size_t *len*, int* *yes*) Function

Sets **yes* to 1 if this address is just the name of a group, 0 otherwise. This is faster than checking if the address has a non-zero length personal, and a zero-length local_part and domain.

yes can be null, though that doesn't serve much purpose other than determining that *no* refers to an address.

Currently, there is no way to determine the end of the group.

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

EINVAL Invalid usage, usually a required argument was null.

ENOENT The index *no* is outside of the range of available addresses.

int **address_to_string** (address_t **addr*, char* *buf*, size_t *len*, size_t* *n*) Function

Returns the entire address list as a single RFC822 formatted address list.

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

EINVAL Invalid usage, usually a required argument was null.

ENOMEM Not enough memory to allocate resources.

int **address_get_count** (address_t *addr*, size_t* *count*) Function

Returns a count of the addresses in the address list.

If *addr* is null, the count is 0. If *count* is not null, the count will be written to **count*.

The return value is 0.

3.13 Example

```
#include <stdio.h>
#include <errno.h>
#include <mailutils/address.h>

#define EPARSE ENOENT

static const char* err_name(int e)
{
    struct {
int e;
const char* s;
    } map[] = {
#define E(e) { e, #e },
E(ENOENT)
E(EINVAL)
E(ENOMEM)
#undef E
{ 0, NULL }
    };
    static char s[sizeof(int) * 8 + 3];
    int i;

    for(i = 0; map[i].s; i++) {
if(map[i].e == e)
    return map[i].s;
    }
    sprintf(s, "[%d]", e);

    return s;
}

static int parse(const char* str)
{
    size_t no = 0;
    size_t pcount = 0;
    int status;

    char buf[BUFSIZ];

    address_t  address = NULL;

    status = address_create(&address, str);

    address_get_count(address, &pcount);

    if(status) {
printf("%s=> error %s\n\n", str, err_name(status));
```

```
return 0;
    } else {
printf("%s=> pcount %d\n", str, pcount);
    }

    for(no = 1; no <= pcount; no++) {
size_t got = 0;
int isgroup;

address_is_group(address, no, &isgroup);

printf("%d ", no);

if(isgroup) {
  address_get_personal(address, no, buf, sizeof(buf), &got);

  printf("group <%s>\n", buf);
} else {
  address_get_email(address, no, buf, sizeof(buf), 0);

  printf("email <%s>\n", buf);
}

address_get_personal(address, no, buf, sizeof(buf), &got);

if(got && !isgroup) printf("   personal <%s>\n", buf);

address_get_comments(address, no, buf, sizeof(buf), &got);

if(got) printf("   comments <%s>\n", buf);

address_get_local_part(address, no, buf, sizeof(buf), &got);

if(got) {
    printf("   local-part <%s>", buf);

    address_get_domain(address, no, buf, sizeof(buf), &got);

    if(got) printf(" domain <%s>", buf);

    printf("\n");
}

address_get_route(address, no, buf, sizeof(buf), &got);

if(got) printf("   route <%s>\n", buf);
    }
    address_destroy(&address);
```

```
    printf("\n");

    return 0;
}

static int parseinput(void)
{
    char buf[BUFSIZ];

    while(fgets(buf, sizeof(buf), stdin) != 0) {
buf[strlen(buf) - 1] = 0;
parse(buf);
    }

    return 0;
}

int main(int argc, const char *argv[])
{
    argc = 1;

    if(!argv[argc]) {
return parseinput();
    }
    for(; argv[argc]; argc++) {
if(strcmp(argv[argc], "-") == 0) {
    parseinput();
} else {
    parse(argv[argc]);
}
    }

    return 0;
}
```

3.14 Locker

```
/* Prefix locker_ is reserved */
#include <mailutils/locker.h>
```

int **locker_create** (locker_t * *plocker*, char **filename*, size_t *len*, int *flags*) — Function

void **locker_destroy** (locker_t * *plocker*) — Function

int **locker_lock** (locker_t *locker*, int *flag*) — Function

`MU_LOCKER_RDLOCK`

`MU_LOCKER_WRLOCK`

```
MU_LOCKER_PID
MU_LOCKER_FCNTL
MU_LOCKER_TIME
```

`int` **locker_touchlock** (`locker_t` *locker*) Function

`int` **locker_unlock** (`locker_t` *locker*) Function

3.15 URL

A mailbox or a mailer can be described in a URL, the string will contain the necessary information to initialize `mailbox_t`, or `mailer_t` properly.

3.15.1 POP3

The POP URL scheme contains a POP server, optional port number and the authentication mechanism. The general form is

```
pop://[<user>[;AUTH=<auth>]@]<host>[:<port>]
 or
pop://[<user>[:<passwd>]@]<host>[:<port>]
```

If *:port* is omitted the default value is 110. Different forms of authentication can be specified with *;AUTH=type*. The special string *;AUTH=** indicates that the client will use a default scheme base on the capability of the server.

```
pop://obelix@gaulois.org
pop://asterix;AUTH=*@france.com
pop://falbala;AUTH=+APOP@france.com
pop://obelix;AUTH=+APOP@village.gaulois.org:2000
pop://obelix:menhir@village.gaulois.org:2000
```

For more complete information see *rfc2368*.

3.15.2 IMAP

The IMAP URL scheme contains an IMAP server, optional port number and the authentication mechanism. The general form is

```
imap://[<user>[;AUTH=<type>]]@<host>[:port][/<mailbox>]
 or
imap://[<user>[:<passwd>]]@<host>[:port][/<mailbox>]
```

If *:port* is omitted the default value is 220. Different forms of authentication can be specified with *;AUTH=type*. The special string *;AUTH=** indicates that the client will use a default scheme base on the capability of the server.

```
imap://obelix@imap.gaulois.org
imap://asterix;AUTH=*@imap.france.com
imap://asterix:potion@imap.france.com
```

For more complete information see *rfc2192*.

3.15.3 File

Local folder should be handle by this URL. It is preferable to let the mailbox recognize the type of mailbox and take the appropriate action.

```
file://path
file://var/mail/user
file://home/obelix/Mail
```

For MMDF, MH local mailboxes URLs are provided, but it is preferable to use `file://path` and let the library figure out which one.

```
mmdf://path
mh://path
```

3.15.4 Mailto

After setting a mailer, `mailto:` is used to tell the mailer where and to whom the message is for.

```
mailto://hostname
```

Mailto can be used to generate short messages, for example to subscribe to mailing lists.

```
mailto://bug-mailutils@gnu.org?body=subscribe
mailto://bug-mailutils@gnu.org?Subject=hello&body=subscribe
```

For more complete information see *rfc2368*.

3.15.5 URL functions

Helper functions are provided to retrieve and set the *URL* fields.

`int` **url_create** (`url_t *`*url*, `const char *`*name*) — Function

Create and the *url* data structure, but do not parse it.

`void` **url_destroy** (`url_t *`) — Function

Destroy the url and free it's resources.

`int` **url_parse** (`url_t` *url*) — Function

Parses the url, after calling this the get functions can be called.

The syntax, condensed from RFC 1738, and extended with the ;auth= of RFC 2384 (for POP) and RFC 2192 (for IMAP) is:

```
url =
    scheme ":" = "//"

    [ user [ ( ":" password ) | ( ";auth=" auth ) ] "¨ ]

    host [ ":" port ]

    [ ( "/" urlpath ) | ( "?" query ) ]
```

This is a generalized URL syntax, and may not be exactly appropriate for any particular scheme.

const **char*** url_to_string (const url_t *url*) Function

int **url_get_scheme** (const url_t *url*, char **schem*, size_t *len*, size_t **n*) Function

int **url_get_user** (const url_t *url*, char **usr*, size_t *len*, size_t **n*) Function

int **url_get_passwd** (const url_t *url*, char **passwd*, size_t *len*, size_t **n*) Function

int **url_get_host** (const url_t *url*, char **host*, size_t *len*, size_t **n*) Function

int **url_get_port** (const url_t *url*, long **port*) Function

int **url_get_path** (const url_t *url*, char **path*, size_t *len*, size_t **n*) Function

int **url_get_query** (const url_t *url*, char **query*, size_tlen, size_t **n*) Function

char* **url_decode** (const char* s) Function

Decodes an RFC 1738 % encoded string, returning the decoded string in allocated memory. If the string is not encoded, this degenerates to a strdup().

3.16 Example

```
#include <mailutils/url.h>
#include <stdio.h>
#include <string.h>

int
main ()
{
  char str[1024];
  char buffer[1024];
  long port = 0;
  int len = sizeof (buffer);
  url_t u = NULL;

  while (fgets (str, sizeof (str), stdin) != NULL)
    {
      int rc;

      str[strlen (str) - 1] = '\0'; /* chop newline */
      if(strspn(str, " \t") == strlen(str))
        continue; /* skip empty lines */
      if ((rc = url_create(&u, str)) != 0)
```

```
        {
          printf(stderr, "url_create %s ERROR: [%d] %s",
            str, rc, strerror(rc));
          exit (1);
        }
        if ((rc = url_parse (u)) != 0)
          {
            printf ("%s --> FAILED: [%d] %s\n",
              str, rc, strerror(rc));
            continue;
          }
        printf ("%s --> SUCCESS\n", str);

        url_get_scheme (u, buffer, len, NULL);
        printf (" scheme <%s>\n", buffer);

        url_get_user (u, buffer, len, NULL);
        printf (" user <%s>\n", buffer);

        url_get_passwd (u, buffer, len, NULL);
        printf (" passwd <%s>\n", buffer);

        url_get_auth (u, buffer, len, NULL);
        printf (" auth <%s>\n", buffer);

        url_get_host (u, buffer, len, NULL);
        printf (" host <%s>\n", buffer);

        url_get_port (u, &port);
        printf (" port %ld\n", port);

        url_get_path (u, buffer, len, NULL);
        printf (" path <%s>\n", buffer);

        url_get_query (u, buffer, len, NULL);
        printf (" query <%s>\n", buffer);

        url_destroy (&u);

      }
    return 0;
  }
```

4 Programs

GNU Mailutils provides a set of programs for handling the email.

4.1 IMAP4 daemon

imap4d has two operation modes:

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

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

This is the default operation mode.

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

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.

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

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

'`-p NUMBER`'
'`--port NUMBER`'
: Listen on given port NUMBER. This option is meaningful only in standalone mode. It defaults to port 143.

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

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

4.2 POP3 daemon

The '`pop3d`' daemon implements the Post Office Protocol server.

pop3d has two operation modes:

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

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

This is the default operation mode.

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

Command line options

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

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

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

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

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

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

4.3 frm — List headers from a mailbox.

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

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

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

'-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. Currently is not implemented. Included for compatibility with '`frm`' program from Elm package.

'-v'
'--version'
: Display version information and exit

4.4 mail — send and receive mail.

`Mail` is an enhanced version of standard '`/bin/mail`' program. As well as its predecessor, it can be used either in sending mode or in reading mode. `Mail` enters sending mode when one or more email addresses were specified in this command line. In this mode the program waits until user finishes composing the message, then attempts to send it to the specified addresses and exits. See See Section 4.4.3 [Composing Mail], page 61, for a detailed description of this behavior.

If the command line contained no email addresses, `mail` switches to reading mode. In this mode it allows to read and manipulate the contents of a mailbox. The URL of the mailbox to operate upon is taken from the argument of '`--file`' command line option. If it is not specified, the user's system mailbox is assumed. See Section 4.4.4 [Reading Mail], page 64 for more detail.

4.4.1 Command line options

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 following command line options:

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

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

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

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

'-i'
'--ignore'
Ignore interrupts.

'-n'
'--norc' Do not read the system-wide mailrc file. See Section 4.4.6 [Mail Configuration Files], page 76.

'-N'
'--nosum' Do not display initial header summary.

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

```
print *
quit
```

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

'-s SUBJ'
'--subject=SUBJ'
Send a message with a Subject of SUBJ. Valid only in sending mode.

'-t'
'--to' Switch to sending mode.

'-u USER'
'--user=USER'
Operate on USER's mailbox. This is equivalent to:

`mail -f/spool_path/USER`

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

`-?`
`--help` Display a help message.

`--usage` Display a short usage summary.

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

4.4.2 How to specify message sets

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

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

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

* Selects all messages in the mailbox.

^ Selects first non-deleted message.

$ Selects last non-deleted message.

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

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

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

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

`:d` Selects all deleted messages.

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

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

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

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

`:t` Selects all tagged messages (see Section 4.4.4.8 [Marking Messages], page 67).

'`:T`' Selects all untagged messages.

Header match

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

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

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

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

Message body match

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

`:/`*string*`/`

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

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

The following are the examples of valid message lists:

4.4.3 Composing mail

You can compose the message by simply typing the contents of it, line by line. But usually this is not enough, you would need to edit your text, to quote some messages, etc. '`Mail`' provides these capabilities through *compose escapes*. The *compose escapes* are single-character commands, preceded by special *escape character*, which defaults to '`~`'. The combination `escape character + command` is recognized as a compose escape only if it occurs at the beginning of a line. If the escape character must appear at the beginning of a line, enter it twice. The actual escape character may be changed by setting the value of `escape` mail variable (see Section 4.4.5 [Mail Variables], page 72).

4.4.3.1 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.

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.

4.4.3.2 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.

4.4.3.3 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.

Please note, that both commands allow you to edit the body of the message, and not its headers. To change message headers, see Section 4.4.3.4 [Modifying the Headers], page 62.

4.4.3.4 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.

4.4.3.5 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 (see Section 4.4.4.5 [Controlling Header Display], page 65).

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

4.4.3.6 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
```

4.4.3.7 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.

4.4.3.8 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.

4.4.3.9 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 (see Section 4.4.4.5 [Controlling Header Display], page 65).

4.4.3.10 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.

4.4.3.11 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 (see Section 4.4.4.14 [Replying], page 71) 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`.

4.4.3.12 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.

4.4.4 Reading mail

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

`mail`
: To read messages from your system mailbox.

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

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

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

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

Unless you have started mail with '`--norc`' command line option, it will read the contents of the system-wide configuration file. Then it reads the contents of user configuration file, if any. See Section 4.4.6 [Mail Configuration Files], page 76 for detailed description of these files. After this initial setup, '`mail`' displays the first page of header lines and enters interactive mode. In interactive mode, '`mail`' displays its prompt ('`?`', if not set otherwise) and executes the commands the user enters.

4.4.4.1 Quitting the program

Following commands quit the program:

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

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

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

4.4.4.2 Obtaining online help

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

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

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

'version'
've' Display program version.

'warranty'
'wa' Display program warranty statement.

4.4.4.3 Moving within a mailbox

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

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

4.4.4.4 Changing mailbox/directory

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

'file [*mailbox*]'
'fi [*mailbox*]'
'folder [*mailbox*]'
'fold [*mailbox*]'
Read in the contents of the specified *mailbox*. The current mailbox is updated as if `quit` command has been issued. If *mailbox* is omitted, the command prints the current mailbox name followed by the summary information regarding it, e.g.:

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

4.4.4.5 Controlling header display

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

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

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

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

4.4.4.6 Displaying information

'`=`' Displays the current message number.

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

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

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

```
& z +2
```

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

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

'`folders`' Displays the value of `folder` variable.

'`summary`'
'`su`' Displays current mailbox summary. E.g.:

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

4.4.4.7 Displaying messages

'`print` [*msglist*]'
'`p` [*msglist*]'

'`type` [*msglist*]'
'`t` [*msglist*]'
: Prints out the messages from *msglist*. If the variable `crt` is set and the number of lines in the message is greater than the number of lines on screen, the message is piped through pager command specified by environment variable `PAGER`. The number of lines on screen is controlled by `screen` variable.

'Print [*msglist*]'
'P [*msglist*]'
'Type [*msglist*]'
'T [*msglist*]'

Like print but also prints out ignored header fields. See Section 4.4.4.5 [Controlling Header Display], page 65 for detailed description of ignored header lists.

'decode [*msglist*]'
'dec [*msglist*]'

Print a multipart message. The `decode` command decodes and prints out specified message parts. E.g.

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

'top [*msglist*]'
'to [*msglist*]'

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

'pipe [*msglist*] [*shell-command*]'
'| [*msglist*] [*shell-command*]'

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

4.4.4.8 Marking messages

'tag [*msglist*]'
'ta [*msglist*]'

Tag messages. The tagged messages can be referred to in message list using ':t' notation.

'untag [*msglist*]'
'unt [*msglist*]'

Clear tags from specified messages. To untag all messages tagged so far type

```
& untag :t
```

'hold [*msglist*]'
'ho [*msglist*]'
'preserve [*msglist*]'
'pre [*msglist*]'

Marks each message to be held in user's system mailbox. This command does not override the effect of `delete` command.

4.4.4.9 Disposing of messages

'`delete [`*msglist*`]`'
'`d [`*msglist*`]`'

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

'`undelete [`*msglist*`]`'
'`u [`*msglist*`]`'

Clear delete mark from the specified messages.

'`dp [`*msglist*`]`'
'`dt [`*msglist*`]`'

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

4.4.4.10 Saving messages

'`save [[`*msglist*`]` *file*`]`'
'`s [[`*msglist*`]` *file*`]`'

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

'`Save [`*msglist*`]`'
'`S [`*msglist*`]`'

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

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

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

'`write [[`*msglist*`]` *file*`]`'
'`w [[`*msglist*`]` *file*`]`'

Similar to `save`, except that only message body (without the header) is saved.

'`Write [`*msglist*`]`'
'`W [`*msglist*`]`'

Similar to `Save`, except that only message body (without the header) is saved.

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

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

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

4.4.4.11 Editing messages

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

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

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

4.4.4.12 Scripting

Comments

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

Displaying arbitrary text

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

Sourcing external command files

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

Setting and unsetting the variables.

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

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

'*name*=*number*'
: Assign a numeric value to the variable.

'*name*'
: Assign boolean `True` value.

'`no`*name*'
: Assign boolean `False` value.

Example:

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

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

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

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

```
& unset askcc crt indentprefix
```

Conditional statements

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

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

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

'`s`'
: True if '`mail`' is operating in mail sending mode.

'`r`'
: True if '`mail`' is operating in mail reading mode.

'`t`'
: True if stdout is a terminal device (as opposed to a regular file).

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

Example:

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

4.4.4.13 Aliasing

'`alias [alias [address...]]`'
'`a [alias [address...]]`'
'`group [alias [address...]]`'
'`g [alias [address...]]`'

With no arguments, prints out all currently-defined aliases. With one argument, prints out that alias. With more than one argument, creates a new alias or changes an old one.

'`unalias [alias...]`'
'`una [alias...]`'

Takes a list of names defined by alias commands and discards the remembered groups of users. The alias names no longer have any significance.

'`alternates name...`'
'`alt name...`'

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

4.4.4.14 Replying

'`mail [address...]`'
'`m [address...]`'

Switches to compose mode. After composing the message, sends messages to the specified addresses.

'`reply [`*msglist*`]`'
'`respond [`*msglist*`]`'
'`r [`*msglist*`]`'

For each message in *msglist*, switches to compose mode and sends the composed message to the sender and all recipients of the message.

'`Reply [`*msglist*`]`'
'`Respond [`*msglist*`]`'
'`R [`*msglist*`]`'

Like `reply`, except that the composed message is sent only to originators of the specified messages.

'`followup [`*msglist*`]`'
'`fo [`*msglist*`]`'

Switches to compose mode. After composing, sends the message to the originators and recipients of all messages in *msglist*.

'`Followup [`*msglist*`]`'
'`F [`*msglist*`]`'

Similar to `followup`, but reply message is sent only to originators of messages in *msglist*.

4.4.4.15 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.

4.4.4.16 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 `!`.

4.4.5 How to alter the behavior of mail

Following variables control the behavior of GNU '`mail`':

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

`appenddeadletter`
: Type: Boolean.
Default: False.
If this variable is `True`, the contents of canceled letter is appended to the user's '`dead.letter`' file. Otherwise it overwrites its contents.

`askbcc`
: Type: Boolean.
Default: False.
When set to `True` the user will be prompted to enter `Bcc` field before composing the message.

`askcc`
: Type: Boolean.
Default: True.
When set to `True` the user will be prompted to enter `Cc` field before composing the message.

`asksub`
: Type: Boolean.
Default: True in interactive mode, False otherwise.
When set to `True` the user will be prompted to enter `Subject` field before composing the message.

`autoinc`
: Type: Boolean.
Default: True.

Automatically incorporate newly arrived messages.

`autoprint`

Type: Boolean.
Default: False.

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

`cmd`

Type: String.
Default: Unset.

Contains default shell command for `pipe`.

`columns`

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

This variable contains the number of columns on terminal screen.

`crt`

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

If `True`, any messages with number of lines greater than number of lines on terminal screen (as determined by `screen` variable) will be printed using program set in the environment variable `ENVIRON`.

`dot`

Type: Boolean.
Default: False.

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

`escape`

Type: String.
Default: `~`

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

`folder`

Type: String.
Default: Unset.

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

`header`

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

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

`hold`

Type: Boolean.
Default: False.

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

`ignore`

Type: Boolean.
Default: False.

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

`ignoreeof`

Type: Boolean.
Default: False.

Controls whether typing EOF character terminates the letter being composed.

`indentprefix`

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

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

`keepsave`

Type: Boolean.
Default: False.

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

`metoo`

Type: Boolean.
Default: False.

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

`mode`

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

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

`noregex`

Type: Boolean.
Default: False.

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

`outfolder`

Type: String.
Default: Unset.

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

`page`

Type: Boolean.
Default: False.

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

`prompt`

Type: String.
Default: `"? "`

Contains the command prompt sequence.

`quiet`

Type: Boolean.
Default: False.

When set, suppresses the output of the startup banner.

`quit`

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

When set, causes keyboard interrupts to terminate the program.

`rc`

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

When this variable is set, '`mail`' will read the system-wide configuration file upon startup. See Section 4.4.6 [Mail Configuration Files], page 76.

`record`

Type: String.
Default: Unset.

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

`save`

Type: Boolean.
Default: True.

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

`screen`

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

This variable contains the number of lines on terminal screen.

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

  Contains the URL of mail transport agent.

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

  Contains the 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.

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

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

`toplines`
: Type: Numeric.
  Default: 5

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

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

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

4.4.6 Personal and system-wide configuration files

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

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

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

4.5 messages — Count the number of messages in a mailbox.

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

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

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

The program accepts following command line options:

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

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

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

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

4.6 readmsg — Extract messages from a folder.

The program is currently in development

4.7 sieve

The program is currently in development

4.8 guimb — A mailbox scanning and processing language.

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

4.8.1 Specifying Scheme Program to Execute

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

'-e'
'--expression EXPR'
: Execute scheme expression.

'-f'
'--file PROGFILE'
: Read and execute program from PROGFILE.

You can specify both of them. In this case, the PROGFILE is executed first.

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

If no mailboxes are specified in the command line, 'guimb' reads and processes its standard input.

4.8.3 Passing Options to Scheme

Sometimes it is necessary to pass some command line options to the scheme procedure. This can be done via '-g' ('--guile-command') command line option. For example:

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

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

```
progfile -opt 24
```

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
```

All arguments enclosed between '-{' and '-}' are not processed by 'guimb', they are instead passed to the Scheme program verbatim.

4.8.4 Guimb Invocation 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.

'-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.

4.8.5 Scheme Procedures and Variables

4.8.5.1 Address Functions

mu-address-get-personal **ADDRESS** *NUM*
Return personal part of an email address.

mu-address-get-comments **ADDRESS** *NUM*

mu-address-get-email **ADDRESS** *NUM*
Return email part of an email address.

mu-address-get-domain **ADDRESS** *NUM*
Return domain part of an email address

mu-address-get-local **ADDRESS** *NUM*
Return local part of an email address.

mu-address-get-count **ADDRESS**
Return number of parts in email address.

4.8.5.2 Mailbox Functions

mu-mailbox-open **URL** *MODE*
Opens a mailbox specified by URL.

mu-mailbox-close **MBOX**
Closes mailbox MBOX

mu-mailbox-get-url **MBOX**
Returns the URL of the mailbox.

MBOX *MODE* mu-mailbox-get-port

Returns a port associated with the contents of the MBOX. MODE is a string defining operation mode of the stream. It may contain any of the two characters: '`r`' for reading, '`w`' for writing.

MBOX *MSGNO* mu-mailbox-get-message

Retrieve from MBOX message # MSGNO.

MBOX mu-mailbox-messages-count

Returns number of messages in the mailbox.

MBOX mu-mailbox-expunge

Expunges deleted messages from the mailbox.

MBOX mu-mailbox-url

Returns the URL of the mailbox

MBOX *MESG* mu-mailbox-append-message

Appends the message to the mailbox

4.8.5.3 Message Functions

MESG mu-message-copy

Creates the copy of the given message.

MESG *HEADER VALUE REPLACE* mu-message-set-header

Sets new VALUE to the header HEADER of the message MESG. If the HEADER is already present in the message its value is replaced with the supplied one if the optional REPLACE is #t. Otherwise new header is created and appended.

MESG mu-message-get-size

Returns the size of the given message.

MESG mu-message-get-lines

Returns number of lines in the given message.

MESG mu-message-get-sender

Returns the sender email address for the message MESG.

MESG *HEADER* mu-message-get-header

Returns the header value of the HEADER in the MESG.

MESG *HEADERS* mu-message-get-header-fields

Returns the list of headers in the MESG. If optional HEADERS is specified it should be a list of header names to restrict return value to.

MESG *LIST REPLACE* mu-message-set-header-fields

Set the headers in the message MESG from LIST LIST is a list of (cons HEADER VALUE) Optional parameter REPLACE specifies whether the new header values should replace the headers already present in the message.

MESG *FLAG* mu-message-delete

Mark given message as deleted. Optional FLAG allows to toggle deleted mark The message is deleted if it is #t and undeleted if it is #f

MESG *FLAG* mu-message-get-flag

Return value of the attribute FLAG.

MESG *FLAG VALUE* mu-message-set-flag

Set the given attribute of the message. If optional VALUE is #f, the attribute is unset.

MESG *FLAG* mu-message-get-user-flag

Returns value of the user attribute FLAG.

MESG *FLAG VALUE* mu-message-set-user-flag

Set the given user attribute of the message. If optional VALUE is #f, the attribute is unset.

MESG *MODE FULL* mu-message-get-port

Returns a port associated with the given MESG. MODE is a string defining operation mode of the stream. It may contain any of the two characters: '`r`' for reading, '`w`' for writing. If optional FULL argument specified, it should be a boolean value. If it is #t then the returned port will allow access to any part of the message (including headers). If it is #f then the port accesses only the message body (the default).

MESG mu-message-get-body

Returns the message body for the message MESG.

MESG *MAILER* mu-message-send

Sends the message MESG. Optional MAILER overrides default mailer settings in mu-mailer.

4.8.5.4 MIME Functions

FLAGS *MESG* mu-mime-create

Creates a new MIME object.

MIME mu-mime-multipart?

Returns #t if MIME is a multipart object.

MIME mu-mime-get-num-parts
Returns number of parts in a MIME object.

MIME *PART* mu-mime-get-part
Returns part number PART from a MIME object.

MIME *MESG* mu-mime-add-part
Adds MESG to the MIME object.

MIME mu-mime-get-message
Converts MIME object to a message.

4.8.5.5 Log Functions

IDENT *OPTION FACILITY* mu-openlog
Opens a connection to the system logger for Guile program.

PRIO *TEXT* mu-logger
Generates a log message to be distributed via syslogd.

Closes *the channel to the system logger open by mu-openlog.* mu-closelog

5 Reporting Bugs

Email bug reports to `bug-mailutils@gnu.org`. Be sure to include the word "mailutils" somewhere in the "Subject:" field.

6 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`.

Concept Index

This is a general index of all issues discussed in this manual

Index

This is an alphabetical list of all MAILUTILS functions.

A

B

C

D

E

F

H

I

L

M

N

P

S

T

U

Table of Contents