Emacs MIME Manual

by Lars Magne Ingebrigtsen

Copyright © 1998,99,2000, 2002 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being none, with the Front-Cover texts being "A GNU Manual", and with the Back-Cover Texts as in (a) below. A copy of the license is included in the section entitled "GNU Free Documentation License" in the Emacs manual.

(a) The FSF's Back-Cover Text is: "You have freedom to copy and modify this GNU Manual, like GNU software. Copies published by the Free Software Foundation raise funds for GNU development."

This document is part of a collection distributed under the GNU Free Documentation License. If you want to distribute this document separately from the collection, you can do so by adding a copy of the license to the document, as described in section 6 of the license.

Emacs MIME

This manual documents the libraries used to compose and display MIME messages.

This is not a manual meant for users; it's a manual directed at people who want to write functions and commands that manipulate MIME elements.

MIME is short for *Multipurpose Internet Mail Extensions*. This standard is documented in a number of RFCs; mainly RFC2045 (Format of Internet Message Bodies), RFC2046 (Media Types), RFC2047 (Message Header Extensions for Non-ASCII Text), RFC2048 (Registration Procedures), RFC2049 (Conformance Criteria and Examples). It is highly recommended that anyone who intends writing MIME-compliant software read at least RFC2045 and RFC2047.

1 Interface Functions

The `mail-parse` library is an abstraction over the actual low-level libraries that are described in the next chapter.

Standards change, and so programs have to change to fit in the new mold. For instance, RFC2045 describes a syntax for the `Content-Type` header that only allows ASCII characters in the parameter list. RFC2231 expands on RFC2045 syntax to provide a scheme for continuation headers and non-ASCII characters.

The traditional way to deal with this is just to update the library functions to parse the new syntax. However, this is sometimes the wrong thing to do. In some instances it may be vital to be able to understand both the old syntax as well as the new syntax, and if there is only one library, one must choose between the old version of the library and the new version of the library.

The Emacs MIME library takes a different tack. It defines a series of low-level libraries (‘`rfc2047.el`’, ‘`rfc2231.el`’ and so on) that parses strictly according to the corresponding standard. However, normal programs would not use the functions provided by these libraries directly, but instead use the functions provided by the `mail-parse` library. The functions in this library are just aliases to the corresponding functions in the latest low-level libraries. Using this scheme, programs get a consistent interface they can use, and library developers are free to create write code that handles new standards.

The following functions are defined by this library:

mail-header-parse-content-type *string* Function

Parse *string*, a `Content-Type` header, and return a content-type list in the following format:

```
("type/subtype"
 (attribute1 . value1)
 (attribute2 . value2)
 ...)
```

Here’s an example:

```
(mail-header-parse-content-type
 "image/gif; name=\"b980912.gif\"")
⇒ ("image/gif" (name . "b980912.gif"))
```

mail-header-parse-content-disposition *string* Function

Parse *string*, a `Content-Disposition` header, and return a content-type list in the format above.

mail-content-type-get *ct attribute* Function

Returns the value of the given *attribute* from the content-type list *ct*.

```
(mail-content-type-get
 '("image/gif" (name . "b980912.gif")) 'name)
⇒ "b980912.gif"
```

mail-header-encode-parameter *param value* Function

Takes a parameter string '*param*=*value*' and returns an encoded version of it. This is used for parameters in headers like '`Content-Type`' and '`Content-Disposition`'.

mail-header-remove-comments *string* Function

Return a comment-free version of *string*.

```
(mail-header-remove-comments
 "Gnus/5.070027 (Pterodactyl Gnus v0.27) (Finnish Landrace)")
⇒ "Gnus/5.070027  "
```

mail-header-remove-whitespace *string* Function

Remove linear white space from *string*. Space inside quoted strings and comments is preserved.

```
(mail-header-remove-whitespace
 "image/gif; name=\"Name with spaces\"")
⇒ "image/gif;name=\"Name with spaces\""
```

mail-header-get-comment *string* Function

Return the last comment in *string*.

```
(mail-header-get-comment
 "Gnus/5.070027 (Pterodactyl Gnus v0.27) (Finnish Landrace)")
⇒ "Finnish Landrace"
```

mail-header-parse-address *string* Function

Parse an address string *string* and return a list containing the mailbox and the plain-text name.

```
(mail-header-parse-address
 "Hrvoje Niksic <hniksic@srce.hr>")
⇒ ("hniksic@srce.hr" . "Hrvoje Niksic")
```

mail-header-parse-addresses *string* Function

Parse *string* as a list of addresses and return a list of elements like the one described above.

```
(mail-header-parse-addresses
 "Hrvoje Niksic <hniksic@srce.hr>, Steinar Bang <sb@metis.no>")
⇒ (("hniksic@srce.hr" . "Hrvoje Niksic")
     ("sb@metis.no" . "Steinar Bang"))
```

mail-header-parse-date *string* Function

Parse a date *string* and return an Emacs time structure.

mail-narrow-to-head Function

Narrow the buffer to the header section of the buffer. Point is placed at the beginning of the narrowed buffer.

mail-header-narrow-to-field Function

Narrow the buffer to the header under point.

mail-encode-encoded-word-region *start end* Function

Encode the non-ASCII words in the region *start*to *end*. For instance, '`Nave`' is encoded as '`=?iso-8859-1?q?Na=EFve?=`'.

mail-encode-encoded-word-buffer Function

Encode the non-ASCII words in the current buffer. This function is meant to be called with the buffer narrowed to the headers of a message.

mail-encode-encoded-word-string *string* Function

Encode the words that need encoding in *string*, and return the result.

```
(mail-encode-encoded-word-string
 "This is nave, baby")
⇒ "This is =?iso-8859-1?q?na=EFve,?= baby"
```

mail-decode-encoded-word-region *start end* Function

Decode the encoded words in the region *start*to *end*.

mail-decode-encoded-word-string *string* Function

Decode the encoded words in *string* and return the result.

```
(mail-decode-encoded-word-string
 "This is =?iso-8859-1?q?na=EFve,?= baby")
⇒ "This is nave, baby"
```

Currently, `mail-parse` is an abstraction over `ietf-drums`, `rfc2047`, `rfc2045` and `rfc2231`. These are documented in the subsequent sections.

2 Basic Functions

This chapter describes the basic, ground-level functions for parsing and handling. Covered here is parsing `From` lines, removing comments from header lines, decoding encoded words, parsing date headers and so on. High-level functionality is dealt with in the next chapter (see Chapter 3 [Decoding and Viewing], page 15).

2.1 rfc2045

RFC2045 is the "main" MIME document, and as such, one would imagine that there would be a lot to implement. But there isn't, since most of the implementation details are delegated to the subsequent RFCs.

So '`rfc2045.el`' has only a single function:

rfc2045-encode-string *parameter value* Function

Takes a *parameter* and a *value* and returns a '*param*=*value*' string. *value* will be quoted if there are non-safe characters in it.

2.2 rfc2231

RFC2231 defines a syntax for the '`Content-Type`' and '`Content-Disposition`' headers. Its snappy name is *MIME Parameter Value and Encoded Word Extensions: Character Sets, Languages, and Continuations*.

In short, these headers look something like this:

```
Content-Type: application/x-stuff;
 title*0*=us-ascii'en'This%20is%20even%20more%20;
 title*1*=%2A%2A%2Afun%2A%2A%2A%20;
 title*2="isn't it!"
```

They usually aren't this bad, though.

The following functions are defined by this library:

rfc2231-parse-string *string* Function

Parse a '`Content-Type`' header *string* and return a list describing its elements.

```
(rfc2231-parse-string
 "application/x-stuff;
 title*0*=us-ascii'en'This%20is%20even%20more%20;
 title*1*=%2A%2A%2Afun%2A%2A%2A%20;
 title*2=\"isn't it!\"")
⇒ ("application/x-stuff"
    (title . "This is even more ***fun*** isn't it!"))
```

rfc2231-get-value *ct attribute* Function

Takes a list *ct* of the format above and returns the value of the specified *attribute*.

rfc2231-encode-string *parameter value* Function

Encode the string '*parameter*=*value*' for inclusion in headers likes '`Content-Type`' and '`Content-Disposition`'.

2.3 ietf-drums

drums is an IETF working group that is working on the replacement for RFC822.

The functions provided by this library include:

ietf-drums-remove-comments *string* Function
: Remove the comments from *string* and return the result.

ietf-drums-remove-whitespace *string* Function
: Remove linear white space from *string* and return the result. Spaces inside quoted strings and comments are left untouched.

ietf-drums-get-comment *string* Function
: Return the last most comment from *string*.

ietf-drums-parse-address *string* Function
: Parse an address *string* and return a list of the mailbox and the plain text name.

ietf-drums-parse-addresses *string* Function
: Parse *string*, containing any number of comma-separated addresses, and return a list of mailbox/plain text pairs.

ietf-drums-parse-date *string* Function
: Parse the date *string* and return an Emacs time structure.

ietf-drums-narrow-to-header Function
: Narrow the buffer to the header section of the current buffer.

2.4 rfc2047

RFC2047 (Message Header Extensions for Non-ASCII Text) specifies how non-ASCII text in headers are to be encoded. This is actually rather complicated, so a number of variables are necessary to tweak what this library does.

The following variables are tweakable:

rfc2047-default-charset Variable
: Characters in this charset should not be decoded by this library. This defaults to '`iso-8859-1`'.

rfc2047-header-encoding-list Variable
: This is an alist of header / encoding-type pairs. Its main purpose is to prevent encoding of certain headers.

The keys can either be header regexps, or `t`.

The values can be either `nil`, in which case the header(s) in question won't be encoded, or `mime`, which means that they will be encoded.

rfc2047-charset-encoding-alist Variable

RFC2047 specifies two forms of encoding—`Q` (a Quoted-Printable-like encoding) and `B` (base64). This alist specifies which charset should use which encoding.

rfc2047-encoding-function-alist Variable

This is an alist of encoding / function pairs. The encodings are `Q`, `B` and `nil`.

rfc2047-q-encoding-alist Variable

The `Q` encoding isn't quite the same for all headers. Some headers allow a narrower range of characters, and that is what this variable is for. It's an alist of header regexps and allowable character ranges.

rfc2047-encoded-word-regexp Variable

When decoding words, this library looks for matches to this regexp.

Those were the variables, and these are the functions:

rfc2047-narrow-to-field Function

Narrow the buffer to the header on the current line.

rfc2047-encode-message-header Function

Should be called narrowed to the header of a message. Encodes according to `rfc2047-header-encoding-alist`.

rfc2047-encode-region *start end* Function

Encodes all encodable words in the region *start* to *end*.

rfc2047-encode-string *string* Function

Encode *string* and return the result.

rfc2047-decode-region *start end* Function

Decode the encoded words in the region *start* to *end*.

rfc2047-decode-string *string* Function

Decode *string* and return the result.

2.5 time-date

While not really a part of the MIME library, it is convenient to document this library here. It deals with parsing '`Date`' headers and manipulating time. (Not by using tesseracts, though, I'm sorry to say.)

These functions convert between five formats: a date string, an Emacs time structure, a decoded time list, a number of seconds, and a day number.

The functions have quite self-explanatory names, so the following just gives an overview of which functions are available.

```
(parse-time-string "Sat Sep 12 12:21:54 1998 +0200")
⇒ (54 21 12 12 9 1998 6 nil 7200)

(date-to-time "Sat Sep 12 12:21:54 1998 +0200")
⇒ (13818 19266)

(time-to-seconds '(13818 19266))
⇒ 905595714.0

(seconds-to-time 905595714.0)
⇒ (13818 19266 0)

(time-to-day '(13818 19266))
⇒ 729644

(days-to-time 729644)
⇒ (961933 65536)

(time-since '(13818 19266))
⇒ (0 430)

(time-less-p '(13818 19266) '(13818 19145))
⇒ nil

(subtract-time '(13818 19266) '(13818 19145))
⇒ (0 121)

(days-between "Sat Sep 12 12:21:54 1998 +0200"
              "Sat Sep 07 12:21:54 1998 +0200")
⇒ 5

(date-leap-year-p 2000)
⇒ t

(time-to-day-in-year '(13818 19266))
⇒ 255
```

And finally, we have `safe-date-to-time`, which does the same as `date-to-time`, but returns a zero time if the date is syntactically malformed.

2.6 qp

This library deals with decoding and encoding Quoted-Printable text.

Very briefly explained, QP encoding means translating all 8-bit characters (and lots of control characters) into things that look like '`=EF`'; that is, an equal sign followed by the byte encoded as a hex string. It is defined in RFC 2045.

The following functions are defined by the library:

quoted-printable-decode-region *from to* &optional *coding-system* Command

QP-decode all the encoded text in the region. If *coding-system* is non-nil, decode bytes into characters with that coding-system. It is probably better not to use *coding-system*; instead decode into a unibyte buffer, decode that appropriately and then interpret it as multibyte.

quoted-printable-decode-string *string* &optional *coding-system* Function

Return a QP-encoded copy of *string*. If *coding-system* is non-nil, decode bytes into characters with that coding-system.

quoted-printable-encode-region *from to* &optional *fold class* Command

QP-encode all the region. If *fold* is non-*nil*, fold lines at 76 characters, as required by the RFC. If *class* is non-`nil`, translate the characters not matched by that regexp class, which should be in the form expected by *skip-chars-forward* and should probably not contain literal eight-bit characters. Specifying *class* makes sense to do extra encoding in header fields.

If variable *mm-use-ultra-safe-encoding* is defined and non-`nil`, fold lines unconditionally and encode '`From `' and '`-`' at the start of lines..

quoted-printable-encode-string *string* Function

Return a QP-encoded copy of *string*.

2.7 base64

Base64 is an encoding that encodes three bytes into four characters, thereby increasing the size by about 33%. The alphabet used for encoding is very resistant to mangling during transit. See section "Base 64 Encoding" in *The Emacs Lisp Reference Manual*.

2.8 binhex

Binhex is an encoding that originated in Macintosh environments. The following function is supplied to deal with these:

binhex-decode-region *start end* &optional *header-only* Function

Decode the encoded text in the region *start* to *end*. If *header-only* is non-`nil`, only decode the '`binhex`' header and return the file name.

2.9 uudecode

Uuencoding is probably still the most popular encoding of binaries used on Usenet, although Base64 rules the mail world.

The following function is supplied by this package:

uudecode-decode-region *start end* &optional *file-name* Function

Decode the text in the region *start* to *end*. If *file-name* is non-`nil`, save the result to *file-name*.

2.10 rfc1843

RFC1843 deals with mixing Chinese and ASCII characters in messages. In essence, RFC1843 switches between ASCII and Chinese by doing this:

```
This sentence is in ASCII.
The next sentence is in GB.~{<:Ky2;S{#,NpJ)l6HK!#~}Bye.
```

Simple enough, and widely used in China.

The following functions are available to handle this encoding:

rfc1843-decode-region *start end* Function

Decode HZ-encoded text in the region *start* to *end*.

rfc1843-decode-string *string* Function

Decode the HZ-encoded *string* and return the result.

2.11 mailcap

As specified by RFC 1524, MIME-aware message handlers parse *mailcap* files from a default list, which can be overridden by the `MAILCAP` environment variable. These describe how elements are supposed to be displayed. Here's an example file:

```
image/*; gimp -8 %s
audio/wav; wavplayer %s
```

This says that all image files should be displayed with `gimp`, and that WAVE audio files should be played by `wavplayer`.

The `mailcap` library parses such files, and provides functions for matching types.

mailcap-mime-data Variable

This variable is an alist of alists containing backup viewing rules for MIME types. These are overridden by rules for a type found in mailcap files. The outer alist is keyed on the major content-type and the inner alists are keyed on the minor content-type (which can be a regular expression).

For example:

```
(("application"
  ("octet-stream"
   (viewer . mailcap-save-binary-file)
   (non-viewer . t)
   (type . "application/octet-stream"))
  ("plain"
   (viewer . view-mode)
   (test fboundp 'view-mode)
   (type . "text/plain")))
```

mailcap-default-mime-data User Option

This variable is the default value of `mailcap-mime-data`. It exists to allow setting the value using Custom. It is merged with values from mailcap files by `mailcap-parse-mailcaps`.

Although it is not specified by the RFC, MIME tools normally use a common means of associating file extensions with defualt MIME types in the absence of other information about the type of a file. The information is found in per-user files '`~/.mime.types`' and system '`mime.types`' files found in quasi-standard places. Here is an example:

```
application/x-dvi dvi
audio/mpeg mpga mpega mp2 mp3
image/jpeg jpeg jpg jpe
```

mailcap-mime-extensions Variable

This variable is an alist MIME types keyed by file extensions. This is overridden by entries found in '`mime.types`' files.

mailcap-default-mime-extensions User Option

This variable is the default value of `mailcap-mime-extensions`. It exists to allow setting the value using Custom. It is merged with values from mailcap files by `mailcap-parse-mimetypes`.

Interface functions:

mailcap-parse-mailcaps &optional *path force* Function

Parse all the mailcap files specified in a path string *path* and merge them with the values from `mailcap-mime-data`. Components of *path* are separated by the `path-separator` character appropriate for the system. If *force* is non-`nil`, the files are re-parsed even if they have been parsed already. If *path* is omitted, use the value of environment variable `MAILCAPS` if it is set; otherwise (on GNU and Unix) use the path defined in RFC 1524, plus '`/usr/local/etc/mailcap`'.

mailcap-parse-mimetypes &optional *path force* Function

Parse all the mimetypes specified in a path string *path* and merge them with the values from `mailcap-mime-extensions`. Components of *path* are separated by the `path-separator` character appropriate for the system. If *path* is omitted, use the value of environment variable `MIMETYPES` if set; otherwise use a default path consistent with that used by `mailcap-parse-mailcaps`. If *force* is non-`nil`, the files are re-parsed even if they have been parsed already.

mailcap-mime-info *string* &optional *request* Function

Gets the viewer command for content-type *string*. `nil` is returned if none is found. Expects *string* to be a complete content-type header line.

If *request* is non-`nil` it specifies what information to return. If it is nil or the empty string, the viewer (second field of the mailcap entry) will be returned. If it is a string, then the mailcap field corresponding to that string will be returned ('`print`', '`description`', whatever). If it is a number, all the information for this viewer is returned. If it is `all`, then all possible viewers for this type is returned.

mailcap-mime-types Function

This function returns a list of all the defined media types.

mailcap-extension-to-mime *extension* Function

This function returns the content type defined for a file with the given *extension*.

3 Decoding and Viewing

This chapter deals with decoding and viewing MIME messages on a higher level.

The main idea is to first analyze a MIME article, and then allow other programs to do things based on the list of *handles* that are returned as a result of this analysis.

3.1 Dissection

The `mm-dissect-buffer` is the function responsible for dissecting a MIME article. If given a multipart message, it will recursively descend the message, following the structure, and return a tree of MIME handles that describes the structure of the message.

3.2 Handles

A MIME handle is a list that fully describes a MIME component.

The following macros can be used to access elements from the *handle* argument:

mm-handle-buffer *handle* Macro

Return the buffer that holds the contents of the undecoded MIME part.

mm-handle-type *handle* Macro

Return the parsed '`Content-Type`' of the part.

mm-handle-encoding *handle* Macro

Return the '`Content-Transfer-Encoding`' of the part.

mm-handle-undisplayer *handle* Macro

Return the function that can be used to remove the displayed part (if it has been displayed).

mm-handle-set-undisplayer *handle function* Macro

Set the undisplayer function for the part to function.

mm-handle-disposition Macro

Return the parsed '`Content-Disposition`' of the part.

mm-handle-disposition Macro

Return the description of the part.

mm-get-content-id *id* Macro

Returns the handle(s) referred to by *id*, the '`Content-ID`' of the part.

3.3 Display

Functions for displaying, removing and saving. In the descriptions below, 'the part' means the MIME part represented by the *handle* argument.

mm-display-part *handle* &optional *no-default* Function

Display the part. Return `nil` if the part is removed, `inline` if it is displayed inline or `external` if it is displayed externally. If *no-default* is non-`nil`, the part is not displayed unless the MIME type of *handle* is defined to be displayed inline or there is an display method defined for it; i.e. no default external method will be used.

mm-remove-part *handle* Function

Remove the part if it has been displayed.

mm-inlinable-p *handle* Function

Return non-`nil` if the part can be displayed inline.

mm-automatic-display-p *handle* Function

Return non-`nil` if the user has requested automatic display of the MIME type of the part.

mm-destroy-part *handle* Function

Free all the resources used by the part.

mm-save-part *handle* Function

Save the part to a file. The user is prompted for a file name to use.

mm-pipe-part *handle* Function

Pipe the part through a shell command. The user is prompted for the command to use.

mm-interactively-view-part *handle* Function

Prompt for a mailcap method to use to view the part and display it externally using that method.

3.4 Customization

The display of MIME types may be customized with the following options.

mm-inline-media-tests User Option

This is an alist where the key is a MIME type, the second element is a function to display the part *inline* (i.e., inside Emacs), and the third element is a form to be `eval`ed to say whether the part can be displayed inline.

This variable specifies whether a part *can* be displayed inline, and, if so, how to do it. It does not say whether parts are *actually* displayed inline.

mm-inlined-types User Option

This, on the other hand, says what types are to be displayed inline, if they satisfy the conditions set by the variable above. It's a list of MIME media types.

mm-automatic-display User Option

This is a list of types that are to be displayed "automatically", but only if the above variable allows it. That is, only inlinable parts can be displayed automatically.

mm-attachment-override-types User Option

Some MIME agents create parts that have a content-disposition of '`attachment`'. This variable allows overriding that disposition and displaying the part inline. (Note that the disposition is only overridden if we are able to, and want to, display the part inline.)

mm-discouraged-alternatives User Option

List of MIME types that are discouraged when viewing '`multipart/alternative`'. Viewing agents are supposed to view the last possible part of a message, as that is supposed to be the richest. However, users may prefer other types instead, and this list says what types are most unwanted. If, for instance, '`text/html`' parts are very unwanted, and '`text/richtech`' parts are somewhat unwanted, then the value of this variable should be set to:

```
("text/html" "text/richtext")
```

mm-inline-large-images-p User Option

When displaying inline images that are larger than the window, XEmacs does not enable scrolling, which means that you cannot see the whole image. To prevent this, the library tries to determine the image size before displaying it inline, and if it doesn't fit the window, the library will display it externally (e.g. with '`ImageMagick`' or '`xv`'). Setting this variable to `t` disables this check and makes the library display all inline images as inline, regardless of their size.

mm-inline-override-p User Option

`mm-inlined-types` may include regular expressions, for example to specify that all '`text/.*`' parts be displayed inline. If a user prefers to have a type that matches such a regular expression be treated as an attachment, that can be accomplished by setting this variable to a list containing that type. For example assuming `mm-inlined-types` includes '`text/.*`', then including '`text/html`' in this variable will cause '`text/html`' parts to be treated as attachments.

3.5 New Viewers

Here's an example viewer for displaying '`text/enriched`' inline:

```
(defun mm-display-enriched-inline (handle)
  (let (text)
    (with-temp-buffer
      (mm-insert-part handle)
```

```
          (save-window-excursion
            (enriched-decode (point-min) (point-max))
            (setq text (buffer-string))))
        (mm-insert-inline handle text)))
```

We see that the function takes a MIME handle as its parameter. It then goes to a temporary buffer, inserts the text of the part, does some work on the text, stores the result, goes back to the buffer it was called from and inserts the result.

The two important helper functions here are `mm-insert-part` and `mm-insert-inline`. The first function inserts the text of the handle in the current buffer. It handles charset and/or content transfer decoding. The second function just inserts whatever text you tell it to insert, but it also sets things up so that the text can be “undisplayed’ in a convenient manner.

4 Composing

Creating a MIME message is boring and non-trivial. Therefore, a library called `mml` has been defined that parses a language called MML (MIME Meta Language) and generates MIME messages.

The main interface function is `mml-generate-mime`. It will examine the contents of the current (narrowed-to) buffer and return a string containing the MIME message.

4.1 Simple MML Example

Here's a simple '`multipart/alternative`':

```
<#multipart type=alternative>
This is a plain text part.
<#part type=text/enriched>
<center>This is a centered enriched part</center>
<#/multipart>
```

After running this through `mml-generate-mime`, we get this:

```
Content-Type: multipart/alternative; boundary="=-=-="


--=-=-=


This is a plain text part.

--=-=-=
Content-Type: text/enriched


<center>This is a centered enriched part</center>

--=-=-=--
```

4.2 MML Definition

The MML language is very simple. It looks a bit like an SGML application, but it's not.

The main concept of MML is the *part*. Each part can be of a different type or use a different charset. The way to delineate a part is with a '`<#part ...>`' tag. Multipart parts can be introduced with the '`<#multipart ...>`' tag. Parts are ended by the '`<#/part>`' or '`<#/multipart>`' tags. Parts started with the '`<#part ...>`' tags are also closed by the next open tag.

There's also the '`<#external ...>`' tag. These introduce '`external/message-body`' parts.

Each tag can contain zero or more parameters on the form '`parameter=value`'. The values may be enclosed in quotation marks, but that's not necessary unless the value contains white space. So '`filename=/home/user/#hello$^yes`' is perfectly valid.

The following parameters have meaning in MML; parameters that have no meaning are ignored. The MML parameter names are the same as the MIME parameter names; the things in the parentheses say which header it will be used in.

'`type`' The MIME type of the part ('`Content-Type`').

'`filename`'
Use the contents of the file in the body of the part ('`Content-Disposition`').

'`charset`' The contents of the body of the part are to be encoded in the character set speficied ('`Content-Type`').

'`name`' Might be used to suggest a file name if the part is to be saved to a file ('`Content-Type`').

'`disposition`'
Valid values are '`inline`' and '`attachment`' ('`Content-Disposition`').

'`encoding`'
Valid values are '`7bit`', '`8bit`', '`quoted-printable`' and '`base64`' ('`Content-Transfer-Encoding`').

'`description`'
A description of the part ('`Content-Description`').

'`creation-date`'
RFC822 date when the part was created ('`Content-Disposition`').

'`modification-date`'
RFC822 date when the part was modified ('`Content-Disposition`').

'`read-date`'
RFC822 date when the part was read ('`Content-Disposition`').

'`size`' The size (in octets) of the part ('`Content-Disposition`').

Parameters for '`application/octet-stream`':

'`type`' Type of the part; informal—meant for human readers ('`Content-Type`').

Parameters for '`message/external-body`':

'`access-type`'
A word indicating the supported access mechanism by which the file may be obtained. Values include '`ftp`', '`anon-ftp`', '`tftp`', '`localfile`', and '`mailserver`'. ('`Content-Type`'.)

'`expiration`'
The RFC822 date after which the file may no longer be fetched. ('`Content-Type`'.)

'`size`' The size (in octets) of the file. ('`Content-Type`'.)

'`permission`'
Valid values are '`read`' and '`read-write`' ('`Content-Type`').

4.3 Advanced MML Example

Here's a complex multipart message. It's a '`multipart/mixed`' that contains many parts, one of which is a '`multipart/alternative`'.

```
<#multipart type=mixed>
<#part type=image/jpeg filename=~/rms.jpg disposition=inline>
<#multipart type=alternative>
This is a plain text part.
<#part type=text/enriched name=enriched.txt>
<center>This is a centered enriched part</center>
<#/multipart>
This is a new plain text part.
<#part disposition=attachment>
This plain text part is an attachment.
<#/multipart>
```

And this is the resulting MIME message:

```
Content-Type: multipart/mixed; boundary="=-=-="


--=-=-=



--=-=-=
Content-Type: image/jpeg;
 filename="~/rms.jpg"
Content-Disposition: inline;
 filename="~/rms.jpg"
Content-Transfer-Encoding: base64

```

```
/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAgGBgcGBQgHBwcJCQgKDBQNDAsLDBkSEw8UHRof
Hh0aHBwgJC4nICIsIxwcKDcpLDAxNDQ0Hyc5PTgyPC4zNDL/wAALCAAwADABAREA/8QAHwAA
AQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQR
BRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6Q0RF
RkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ip
qrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3+Pn6/9oACAEB
AAA/AO/rifFHjldNuGsrDa0qcSSHkA+gHrXKw+LtWLrMb+RgTyhbr+HSug07xNqV9fQtZrNI
AyiaE/NuBPOOP0rvRNE880KOC8TbXXGCv1FPqjrF4LDR7u5L7SkTFT/ALWOP1xXgTuXfc7E
sx6nua6rwp4IvvEM8chCxWx0dzn7wz6V9AaB4SO7w9p5itow0rDLSY5Pt9K43x066P4xs71m
2QXiGCbA4yOVJ9+1aYORkdK434lyNH4ahCnG66VT9Nj15JFbPdX0MS43M4VQf5/yr2vSpLnw
5ZW8dlCZ8KFXjOPX0/mK6rSPEGt3Angu44fNEReHYNvIH3TzXDeKNO8RX+kSX2ouZkicTIOc
L+g7E810ulFjpVtv3bwgB3HJyK5L4quY/C9sVxk3ij/xx6850u7t1mtp/wDlpEw3An3Jr3Dw
34gsbWza4nBlhC5LDsaW6+IFgupQyCF3iHH7gA7c9R9ay7zx6t7aX9jHC4smhfBkGCvHGfrm
tLQ7hbnRrV1GPkAP1x1/Hr+Ncr8Vzjwrbf8AX6v/AKA9eQRyYlQk8Yx9K6XTNbkgia2ciSIn
7p5Ga9Atte0LTLK06it4i7dVRFJDcZ4PvXN+JvEMF9bILVGXJLSZ4zkjivRPDaeX4b08HOTC
p0ffmua+KkbS+GLVUGT9tT/0B68eeIpIFYjB70+O0VXyoOM9+M1eaWeCLzHPyHGO/NVWvJJm
jQ8KGH1NfQWhXXSXmh2c8eArRLwO3HSv/2Q==
```

```
--=-=-=
Content-Type: multipart/alternative; boundary="==-=-="


--==-=-=


This is a plain text part.

--==-=-=
Content-Type: text/enriched;
 name="enriched.txt"


<center>This is a centered enriched part</center>

--==-=-=--

--=-=-=

This is a new plain text part.

--=-=-=
Content-Disposition: attachment


This plain text part is an attachment.

--=-=-=--
```

4.4 Charset Translation

During translation from MML to MIME, for each MIME part which has been composed inside Emacs, an appropriate MIME charset has to be chosen.

If you are running a non-Mule XEmacs, or Emacs in unibyte mode[1], this process is simple: if the part contains any non-ASCII (8-bit) characters, the MIME charset given by `mail-parse-charset` (a symbol) is used. (Never set this variable directly, though. If you want to change the default charset, please consult the documentation of the package which you use to process MIME messages. See section "Various Message Variables" in *Message Manual*, for example.) If there are only ASCII characters, the MIME charset '`US-ASCII`' is used, of course.

In a normal (multibyte) Emacs session, a list of coding systems is derived that can encode the message part's content and correspond to MIME charsets (according to their `mime-charset` property). This list is according to the normal priority rules and the highest priority one is chosen to encode the part. If no such coding system can encode the part's contents, they are split into several parts such that each can be encoded with an appropriate

[1] Deprecated!

coding system/MIME charset.[2] Note that this procedure works with any correctly-defined coding systems, not just built-in ones. Given a suitably-defined UTF-8 coding system—one capable of encoding the Emacs charsets you use—it is not normally necessary to split a part by charset.

It isn't possible to do this properly in XEmacs/Mule. Instead, a list of the Mule charsets used in the part is obtained, and the corresponding MIME charsets are determined by lookup in `mm-mime-mule-charset-alist`. If the list elements all correspond to a single MIME charset, that is used to encode the part. Otherwise, the part is split as above.

4.5 Conversion

A (multipart) MIME message can be converted to MML with the `mime-to-mml` function. It works on the message in the current buffer, and substitutes MML markup for MIME boundaries. Non-textual parts do not have their contents in the buffer, but instead have the contents in separate buffers that are referred to from the MML tags.

An MML message can be converted back to MIME by the `mml-to-mime` function.

These functions are in certain senses "lossy"—you will not get back an identical message if you run MIME-TO-MML and then MML-TO-MIME. Not only will trivial things like the order of the headers differ, but the contents of the headers may also be different. For instance, the original message may use base64 encoding on text, while MML-TO-MIME may decide to use quoted-printable encoding, and so on.

In essence, however, these two functions should be the inverse of each other. The resulting contents of the message should remain equivalent, if not identical.

[2] The part can only be split at line boundaries, though—if more than one MIME charset is required to encode a single line, it is not possible to encode the part.

5 Standards

The Emacs MIME library implements handling of various elements according to a (somewhat) large number of RFCs, drafts and standards documents. This chapter lists the relevant ones. They can all be fetched from '`http://quimby.gnus.org/notes/`'.

RFC822 *STD11*	Standard for the Format of ARPA Internet Text Messages.
RFC1036	Standard for Interchange of USENET Messages
RFC1524	A User Agent Configuration Mechanism For Multimedia Mail Format Information
RFC2045	Format of Internet Message Bodies
RFC2046	Media Types
RFC2047	Message Header Extensions for Non-ASCII Text
RFC2048	Registration Procedures
RFC2049	Conformance Criteria and Examples
RFC2231	MIME Parameter Value and Encoded Word Extensions: Character Sets, Languages, and Continuations
RFC1843	HZ - A Data Format for Exchanging Files of Arbitrarily Mixed Chinese and ASCII characters
draft-ietf-drums-msg-fmt-05.txt	Draft for the successor of RFC822
RFC2112	The MIME Multipart/Related Content-type
RFC1892	The Multipart/Report Content Type for the Reporting of Mail System Administrative Messages
RFC2183	Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field

6 Index

(Index is nonexistent)

Short Contents

Table of Contents