NAME

Mail::Message::Body − the data of a body in a message

INHERITANCE

```
Mail::Message::Body has extra code in
  Mail::Message::Body::Construct
  Mail::Message::Body::Encode

Mail::Message::Body
  is a Mail::Reporter

Mail::Message::Body is extended by
  Mail::Message::Body::File
  Mail::Message::Body::Lines
  Mail::Message::Body::Multipart
  Mail::Message::Body::Nested
  Mail::Message::Body::String

Mail::Message::Body is realized by
  Mail::Message::Body::Delayed
```

SYNOPSIS

```
my Mail::Message $msg = ...;
my $body  = $msg->body;
my @text  = $body->lines;
my $text  = $body->string;
my $file  = $body->file;  # IO::File
$body->print(\*FILE);

my $content_type = $body->type;
my $transfer_encoding = $body->transferEncoding;
my $encoded = $body->encode(mime_type => 'text/html',
   charset => 'us-ascii', transfer_encoding => 'none');\n";
my $decoded = $body->decoded;
```

DESCRIPTION

The encoding and decoding functionality of a Mail::Message::Body is implemented in the Mail::Message::Body::Encode package. That package is automatically loaded when encoding and decoding of messages needs to take place. Methods to simply build an process body objects are implemented in Mail::Message::Body::Construct.

The body of a message (a Mail::Message object) is stored in one of the many body types. The functionality of each body type is equivalent, but there are performance differences. Each body type has its own documentation with details about its implementation.

Extends ‘‘DESCRIPTION’’ in Mail::Reporter.

OVERLOADED

overload: **“”**

(stringification) Returns the body as string −−which will trigger completion— unless called to produce a string for `Carp`. The latter to avoid deep recursions.

example: stringification of body

```
print $msg->body;   # implicit by print

my $body = $msg->body;
my $x    = "$body"; # explicit by interpolation
```

overload: **'==' and '!='**

(numeric comparison) compares if two references point to the same message. This only produces correct results is both arguments are message references **within the same folder**.

example: use of numeric comparison on a body

```
my $skip = $folder->message(3);
foreach my $msg (@$folder)
{   next if $msg == $skip;
    $msg->send;
}
```

overload: **@{}**

When a body object is used as being an array reference, the lines of the body are returned. This is the same as using **lines()**.

example: using a body as array

```
print $body->lines->[1];  # second line
print $body->[1];         # same

my @lines = $body->lines;
my @lines = @$body;       # same
```

overload: **bool**

Always returns a true value, which is needed to have overloaded objects to be used as in `if($body)`. Otherwise, `if(defined $body)` would be needed to avoid a runtime error.

METHODS

Extends "METHODS" in Mail::Reporter.

Constructors

Extends "Constructors" in Mail::Reporter.

$obj->**clone**()

Return a copy of this body, usually to be included in a cloned message. Use **Mail::Message::clone()** for a whole message.

Mail::Message::Body->**new**(%options)

BE WARNED that, what you specify here are encodings and such which are already in place. The options will not trigger conversions. When you need conversions, first create a body with options which tell what you've got, and then call **encode()** for what you need.

```
-Option           --Defined in     --Default
 based_on                           undef
 charset                            'PERL' or <undef>
 checked                            <false>
 content_id                         undef
 data                               undef
 description                        undef
 disposition                        undef
 eol                                'NATIVE'
 file                               undef
 filename                           undef
 log                Mail::Reporter  'WARNINGS'
 message                            undef
 mime_type                          'text/plain'
 modified                           <false>
 trace              Mail::Reporter  'WARNINGS'
 transfer_encoding                  'none'
```

based_on => BODY
The information about encodings must be taken from the specified BODY, unless specified differently.

charset => CHARSET|'PERL'
Defines the character-set which is used in the data. Only useful in combination with a `mime_type` which refers to `text` in any shape, which does not contain an explicit charset already. This field is case-insensitive.

When a known CHARSET is provided and the mime type says ''text'', then the data is expected to be bytes in that particular encoding (see Encode). When 'PERL' is given, then then the data is in Perl's internal encoding (either latin1 or utf8, you shouldn't know!) More details in ''Character encoding PERL''

checked => BOOLEAN
Whether the added information has been check not to contain illegal octets with respect to the transfer encoding and mime type. If not checked, and then set as body for a message, it will be.

content_id => STRING
In multipart/related MIME content, the content_id is required to allow access to the related content via a cid:<...> descriptor of an inline disposition.

A `Content-ID` is supposed to be globally unique. As such, it is common to append '@computer.domain' to the end of some unique string. As other content in the multipart/related container also needs to know what this `Content-ID` is, this should be left to the imagination of the person making the content (for now).

As a MIME header field, the `Content-ID` string is expected to be inside angle brackets

data => ARRAY-OF-LINES | STRING
The content of the body. The only way to set the content of a body is during the creation of the body. So if you want to modify the content of a message, you need to create a new body with the new content and add that to the body. The reason behind this, is that correct encodings and body information must be guaranteed. It avoids your hassle in calculating the number of lines in the body, and checking whether bad characters are enclosed in text.

Specify a reference to an ARRAY of lines, each terminated by a newline. Or one STRING which may contain multiple lines, separated and terminated by a newline.

description => STRING|FIELD
Informal information about the body content. The data relates to the `Content-Description` field. Specify a STRING which will become the field content, or a real FIELD.

disposition => STRING|FIELD
How this message can be decomposed. The data relates to the `Content-Disposition` field. Specify a STRING which will become the field content, or a real FIELD.

The content of this field is specified in RFC 1806. The body of the field can be`inline`, to indicate that the body is intended to be displayed automatically upon display of the message. Use `attachment` to indicate that they are separate from the main body of the mail message, and that their display should not be automatic, but contingent upon some further action of the user.

The `filename` attribute specifies a name to which is suggested to the reader of the message when it is extracted.

eol => 'CR'|'LF'|'CRLF'|'NATIVE'
Convert the message into having the specified string as line terminator for all lines in the body. `NATIVE` is used to represent the `\n` on the current platform and will be translated in the applicable one.

BE WARNED that folders with a non-native encoding may appear on your platform, for instance in Windows folders handled from a UNIX system. The eol encoding has effect on the size of the body!

file => FILENAME|FILEHANDLE|IOHANDLE
Read the data from the specified file, file handle, or object of type `IO::Handle`.

filename => FILENAME
[3.001] Overrule/set filename for content-disposition

log => LEVEL
message => MESSAGE
The message where this body belongs to.

mime_type => STRING|FIELD|MIME
The type of data which is added. You may specify a content of a header line as STRING, or a FIELD object. You may also specify a MIME::Type object. In any case, it will be kept internally as a real field (a Mail::Message::Field object). This relates to the `Content-Type` header field.

A mime-type specification consists of two parts: a general class (`text`, `image`, `application`, etc) and a specific sub-class. Examples for specific classes with `text` are `plain`, `html`, and `xml`. This field is case-insensitive but case preserving. The default mime-type is `text/plain`,

modified => BOOLEAN
Whether the body is flagged modified, directly from its creation.

trace => LEVEL
transfer_encoding => STRING|FIELD
The encoding that the data has. If the data is to be encoded, than you will have to call **encode()** after the body is created. That will return a new encoded body. This field is case-insensitive and relates to the `Content-Transfer-Encoding` field in the header.

example:

```
my $body = Mail::Message::Body::String->new(file => \*IN,
   mime_type => 'text/html; charset="ISO-8859-1"');

my $body = Mail::Message::Body::Lines->new(data => ['first', $second],
   charset => 'ISO-10646', transfer_encoding => 'none');

my $body = Mail::Message::Body::Lines->new(data => \@lines,
   transfer_encoding => 'base64');

my $body = Mail::Message::Body::Lines->new(file => 'picture.gif',
   mime_type => 'image/gif', content_id => '<12345@example.com>',
   disposition => 'inline');
```

Constructing a body

`$obj`->**attach**($messages, `%options`)
Inherited, see ''Constructing a body'' in Mail::Message::Body::Construct

`$obj`->**check**()
Inherited, see ''Constructing a body'' in Mail::Message::Body::Encode

`$obj`->**concatenate**($components)
Inherited, see ''Constructing a body'' in Mail::Message::Body::Construct

`$obj`->**decoded**(%options)
Returns a body, an object which is (a sub-)class of a Mail::Message::Body, which contains a simplified representation of textual data. The returned object may be the object where this is called on, but may also be a new body of any type.

```
my $dec = $body->decoded;
```

is equivalent with

```
my $dec = $body->encode
  ( mime_type         => 'text/plain'
  , transfer_encoding => 'none'
  , charset           => 'PERL'
  );
```

The `$dec` which is returned is a body. Ask with the**mimeT ype()** method what is produced. This `$dec` body is **not related to a header**.

```
-Option     --Default
 result_type  <same as current>
```

result_type => CLASS

`$obj->`**encode**(%options)
Inherited, see ‘‘Constructing a body’’ in Mail::Message::Body::Encode

`$obj->`**encoded**()
Inherited, see ‘‘Constructing a body’’ in Mail::Message::Body::Encode

`$obj->`**eol**([’CR’|’LF’|’CRLF’|’NATIVE’])
Returns the character (or characters) which are used to separate lines within this body. When a kind of separator is specified, the body is translated to contain the specified line endings.

example:

```
my $body = $msg->decoded->eol('NATIVE');
my $char = $msg->decoded->eol;
```

`$obj->`**foreachLine**(CODE)
Inherited, see ‘‘Constructing a body’’ in Mail::Message::Body::Construct

`$obj->`**stripSignature**(%options)
Inherited, see ‘‘Constructing a body’’ in Mail::Message::Body::Construct

`$obj->`**unify**($body)
Inherited, see ‘‘Constructing a body’’ in Mail::Message::Body::Encode

The body

`$obj->`**isDelayed**()
Returns a true or false value, depending on whether the body of this message has been read from file. This can only false for a Mail::Message::Body::Delayed.

`$obj->`**isMultipart**()
Returns whether this message-body contains parts which are messages by themselves.

`$obj->`**isNested**()
Only true for a message body which contains exactly one sub-message: the `Mail::Message::Body::Nested` body type.

`$obj->`**message**([$message])
Returns the message (or message part) where this body belongs to, optionally setting it to a new `$message` first. If `undef` is passed, the body will be disconnected from the message.

`$obj->`**partNumberOf**($part)
Returns a string for multiparts and nested, otherwise an error. It is used in **Mail::Message::partNumber()**.

About the payload

`$obj->`**charset**()
Returns the character set which is used in the text body as string. This is part of the result of what the `type` method returns.

`$obj->`**checked**([BOOLEAN])
Returns whether the body encoding has been checked or not (optionally after setting the flag to a new value).

`$obj->`**contentId**([STRING|$field])
Returns (optionally after setting) the id (unique reference) of a message part. The related header field is `Content-ID`. A Mail::Message::Field object is returned (which stringifies into the field content). The field content will be `none` if no disposition was specified.

The argument can be a STRING (which is converted into a field), or a fully prepared header `$field`.

`$obj->`**description**([STRING|$field])
Returns (optionally after setting) the informal description of the body content. The related header field is `Content-Description`. A Mail::Message::Field object is returned (which stringifies into the field content). The field content will be `none` if no disposition was specified.

The argument can be a STRING (which is converted into a field), or a fully prepared header field.

`$obj->`**disposition**([STRING|$field])
Returns (optionally after setting) how the message can be disposed (unpacked). The related header field is `Content-Disposition`. A Mail::Message::Field object is returned (which stringifies into the field content). The field content will be `none` if no disposition was specified.

The argument can be a STRING (which is converted into a field), or a fully prepared header field.

`$obj->`**dispositionFilename**([$directory])
Inherited, see ‘‘About the payload’’ in Mail::Message::Body::Encode

`$obj->`**isBinary**()
Inherited, see ‘‘About the payload’’ in Mail::Message::Body::Encode

`$obj->`**isText**()
Inherited, see ‘‘About the payload’’ in Mail::Message::Body::Encode

`$obj->`**mimeType**()
Returns a MIME::Type object which is related to this body’s type. This differs from the `type` method, which results in a Mail::Message::Field.

example:

```
if($body->mimeType eq 'text/html') {...}
print $body->mimeType->simplified;
```

`$obj->`**nrLines**()
Returns the number of lines in the message body. For multi-part messages, this includes the header lines and boundaries of all the parts.

`$obj->`**size**()
The total number of bytes in the message body. The size of the body is computed in the shape it is in. For example, if this is a base64 encoded message, the size of the encoded data is returned; you may want to call **Mail::Message::decoded()** first.

`$obj->`**transferEncoding**([STRING|$field])
Returns the transfer-encoding of the data within this body as Mail::Message::Field (which stringifies to its content). If it needs to be changed, call the **encode()** or **decoded()** method. When no encoding is present, the field contains the text `none`.

The optional STRING or `$field` enforces a new encoding to be set, without the actual required translations.

example:

```
my $transfer = $msg->decoded->transferEncoding;
$transfer->print;   # --> Content-Encoding: base64
print $transfer;    # --> base64

if($msg->body->transferEncoding eq 'none') {...}
```

$obj->**type**([STRING|$field])

Returns the type of information the body contains as Mail::Message::Field object. The type is taken from the header field `Content-Type`. If the header did not contain that field, then you will get a default field containing `text/plain`.

You usually can better use **mimeType()**, because that will return a clever object with type information.

example:

```
my $msg     = $folder->message(6);
$msg->get('Content-Type')->print;
   # --> Content-Type: text/plain; charset="us-ascii"

my $content = $msg->decoded;
my $type    = $content->type;

print "This is a $type message\n";
   # --> This is a text/plain; charset="us-ascii" message

print "This is a ", $type->body, "message\n";
   # --> This is a text/plain message

print "Comment: ", $type->comment, "\n";
   # --> Comment: charset="us-ascii"
```

Access to the payload

$obj->**endsOnNewline**()

Returns whether the last line of the body is terminated by a new-line (in transport it will become a CRLF). An empty body will return true as well: the newline comes from the line before it.

$obj->**file**()

Return the content of the body as a file handle. The returned stream may be a real file, or a simulated file in any form that Perl supports. While you may not be able to write to the file handle, you can read from it.

WARNING: Even if the file handle supports writing, do not write to the file handle. If you do, some of the internal values of the Mail::Message::Body may not be updated.

$obj->**lines**()

Return the content of the body as a list of lines (in LIST context) or a reference to an array of lines (in SCALAR context). In scalar context the array of lines is cached to avoid needless copying and therefore provide much faster access for large messages.

To just get the number of lines in the body, use the **nrLines()** method, which is usually much more efficient.

BE WARNED: For some types of bodies the reference will refer to the original data. You must not change the referenced data! If you do, some of the essential internal variables of the Mail::Message::Body may not be updated.

example:

```
my @lines    = $body->lines;     # copies lines
my $line3    = ($body->lines)[3] # only one copy
print $lines[0];

my $linesref = $body->lines;     # reference to originals
my $line3    = $body->lines->[3] # only one copy (faster)
print $linesref->[0];

print $body->[0];                # by overloading
```

`$obj`->**print**([$fh])
Print the body to the specified `$fh` (defaults to the selected handle). The handle may be a GLOB, an IO::File object, or... any object with a `print()` method will do. Nothing useful is returned.

`$obj`->**printEscapedFrom**($fh)
Print the body to the specified `$fh` but all lines which start with 'From ' (optionally already preceded by >'s) will habe an > added in front. Nothing useful is returned.

`$obj`->**string**()
Return the content of the body as a scalar (a single string). This is a copy of the internally kept information.

example:

```
my $text = $body->string;
print "Body: $body\n";     # by overloading
```

`$obj`->**stripTrailingNewline**()
Remove the newline from the last line, or the last line if it does not contain anything else than a newline.

`$obj`->**write**(%options)
Write the content of the body to a file. Be warned that you may want to decode the body before writing it!

```
-Option  --Default
 filename  <required>
```

filename => FILENAME

example: write the data to a file

```
use File::Temp;
my $fn = tempfile;
$message->decoded->write(filename => $fn)
   or die "Couldn't write to $fn: $!\n";
```

example: using the content-disposition information to write

```
use File::Temp;
my $dir = tempdir; mkdir $dir or die;
my $fn  = $message->body->dispositionFilename($dir);
$message->decoded->write(filename => $fn)
   or die "Couldn't write to $fn: $!\n";
```

Internals

`$obj`->**addTransferEncHandler**(`$name`, <$class|$object>)
Mail::Message::Body->**addTransferEncHandler**(`$name`, <$class|$object>)
Inherited, see ''Internals'' in Mail::Message::Body::Encode

`$obj`->**contentInfoFrom**($head)
Transfer the body related info from the header into this body.

`$obj->`**contentInfoTo**($head)
Copy the content information (the `Content-*` fields) into the specified `$head`. The body was created from raw data without the required information, which must be added. See also **contentInfoFrom()**.

`$obj->`**fileLocation**([$begin, `$end`])
The location of the body in the file. Returned a list containing begin and end. The begin is the offsets of the first byte if the folder used for this body. The end is the offset of the first byte of the next message.

`$obj->`**getTransferEncHandler**($type)
Inherited, see ''Internals'' in Mail::Message::Body::Encode

`$obj->`**isModified**()
Returns whether the body has changed.

`$obj->`**load**()
Be sure that the body is loaded. This returns the loaded body.

`$obj->`**modified**([BOOLEAN])
Change the body modification flag. This will force a re-write of the body to a folder file when it is closed. It is quite dangerous to change the body: the same body may be shared between messages within your program.

Especially be warned that you have to change the message-id when you change the body of the message: no two messages should have the same id.

Without value, the current setting is returned, although you can better use **isModified()**.

`$obj->`**moveLocation**([$distance])
Move the registration of the message to a new location over `$distance`. This is called when the message is written to a new version of the same folder-file.

`$obj->`**read**(`$parser`, `$head`, `$bodytype`, [$chars, [$lines]])
Read the body with the `$parser` from file. The implementation of this method will differ between types of bodies. The `$bodytype` argument is a class name or a code reference of a routine which can produce a class name, and is used in multipart bodies to determine the type of the body for each part.

The `$chars` argument is the estimated number of bytes in the body, or `undef` when this is not known. This data can sometimes be derived from the header (the `Content-Length` line) or file-size.

The second argument is the estimated number of `$lines` of the body. It is less useful than the `$chars` but may be of help determining whether the message separator is trustworthy. This value may be found in the `Lines` field of the header.

Error handling
Extends ''Error handling'' in Mail::Reporter.

`$obj->`**AUTOLOAD**()
When an unknown method is called on a message body object, this may not be problematic. For performance reasons, some methods are implemented in separate files, and only demand-loaded. If this delayed compilation of additional modules does not help, an error will be produced.

`$obj->`**addReport**($object)
Inherited, see ''Error handling'' in Mail::Reporter

`$obj->`**defaultTrace**([$level]|[$loglevel, `$tracelevel`]|[$level, `$callback`])
Mail::Message::Body->**defaultTrace**([$level]|[$loglevel, `$tracelevel`]|[$level, `$callback`])
Inherited, see ''Error handling'' in Mail::Reporter

`$obj->`**errors**()
Inherited, see ‘‘Error handling’’ in Mail::Reporter

`$obj->`**log**([$level, [$strings]])
Mail::Message::Body->**log**([$level, [$strings]])
Inherited, see ‘‘Error handling’’ in Mail::Reporter

`$obj->`**logPriority**($level)
Mail::Message::Body->**logPriority**($level)
Inherited, see ‘‘Error handling’’ in Mail::Reporter

`$obj->`**logSettings**()
Inherited, see ‘‘Error handling’’ in Mail::Reporter

`$obj->`**notImplemented**()
Inherited, see ‘‘Error handling’’ in Mail::Reporter

`$obj->`**report**([$level])
Inherited, see ‘‘Error handling’’ in Mail::Reporter

`$obj->`**reportAll**([$level])
Inherited, see ‘‘Error handling’’ in Mail::Reporter

`$obj->`**trace**([$level])
Inherited, see ‘‘Error handling’’ in Mail::Reporter

`$obj->`**warnings**()
Inherited, see ‘‘Error handling’’ in Mail::Reporter

Cleanup

Extends ‘‘Cleanup’’ in Mail::Reporter.

`$obj->`**DESTROY**()
Inherited, see ‘‘Cleanup’’ in Mail::Reporter

DETAILS

Access to the body

A body can be contained in a message, but may also live without a message. In both cases it stores data, and the same questions can be asked: what type of data it is, how many bytes and lines, what encoding is used. Any body can be encoded and decoded, returning a new body object. However, bodies which are part of a message will always be in a shape that they can be written to a file or send to somewhere: they will be encoded if needed.

. Example

```
my $body    = Mail::Message::Body::String->new(mime_type => 'image/gif');
$body->print(\*OUT);    # this is binary image data...

my $encoded = $message->body($body);
$encoded->print(\*OUT); # ascii data, encoded image
```

Now encoded refers to the body of the `$message` which is the content of `$body` in a shape that it can be transmitted. Usually`base64` encoding is used.

Body class implementation

The body of a message can be stored in many ways. Roughly, the implementations can be split in two groups: the data collectors and the complex bodies. The primer implement various ways to access data, and are full compatible: they only differ in performance and memory footprint under different circumstances. The latter are created to handle complex multiparts and lazy extraction.

Data collector bodies

- Mail::Message::Body::String

  The whole message body is stored in one scalar. Small messages can be contained this way without

performance penalties.

- Mail::Message::Body::Lines

  Each line of the message body is stored as single scalar. This is a useful representation for a detailed look in the message body, which is usually line-organized.

- Mail::Message::Body::File

  The message body is stored in an external temporary file. This type of storage is especially useful when the body is large, the total folder is large, or memory is limited.

- Mail::Message::Body::InFolder

  NOT IMPLEMENTED YET. The message is kept in the folder, and is only taken out when the content is changed.

- Mail::Message::Body::External

  NOT IMPLEMENTED YET. The message is kept in a separate file, usually because the message body is large. The difference with the `::External` object is that this external storage stays this way between closing and opening of a folder. The `::External` object only uses a file when the folder is open.

Complex bodies

- Mail::Message::Body::Delayed

  The message-body is not yet read, but the exact location of the body is known so the message can be read when needed. This is part of the lazy extraction mechanism. Once extracted, the object can become any simple or complex body.

- Mail::Message::Body::Multipart

  The message body contains a set of sub-messages (which can contain multipart bodies themselves). Each sub-message is an instance of Mail::Message::Part, which is an extension of Mail::Message.

- Mail::Message::Body::Nested

  Nested messages, like `message/rfc822`: they contain a message in the body. For most code, they simply behave like multiparts.

Character encoding PERL

A body object can be part of a message, or stand-alone. In case it is a part of a message, the ‘‘transport encoding’’ and the content must be in a shape that the data can be transported via SMTP.

However, when you want to process the body data in simple Perl (or when you construct the body data from normal Perl strings), you need to be aware of Perl’s internal representation of strings. That can either be latin1 or utf8 (not real UTF-8, but something alike, see the perlunicode manual page) So, before you start using the data from an incoming message, do

```
my $body  = $msg->decoded;
my @lines = $body->lines;
```

Now, the body has character-set ’PERL’ (when it is text)

When you create a new body which contains text content (the default), it will be created with character-set ’PERL’ unless you specify a character-set explicitly.

```
my $body = Mail::Box::Body::Lines->new(data => \@lines);
# now mime=text/plain, charset=PERL

my $msg  = Mail::Message->buildFromBody($body);
$msg->body($body);
$msg->attach($body);   # etc
# these all will convert the charset=PERL into real utf-8
```

DIAGNOSTICS

Warning: Charset `$name` is not known

The encoding or decoding of a message body encounters a character set which is not understood by Perl's Encode module.

Warning: No decoder defined for transfer encoding `$name`.

The data (message body) is encoded in a way which is not currently understood, therefore no decoding (or recoding) can take place.

Warning: No encoder defined for transfer encoding `$name`.

The data (message body) has been decoded, but the required encoding is unknown. The decoded data is returned.

Error: Package `$package` does not implement `$method`.

Fatal error: the specific package (or one of its superclasses) does not implement this method where it should. This message means that some other related classes do implement this method however the class at hand does not. Probably you should investigate this and probably inform the author of the package.

Warning: Unknown line terminator `$eol` ignored

SEE ALSO

This module is part of Mail-Message distribution version 3.012, built on February 11, 2022. Website: *http://perl.overmeer.net/CPAN/*

LICENSE