NAME

Mail::Message::Head::Complete − the header of one message

INHERITANCE

```
Mail::Message::Head::Complete
  is a Mail::Message::Head
  is a Mail::Reporter

Mail::Message::Head::Complete is extended by
  Mail::Message::Head::Partial
  Mail::Message::Replace::MailHeader

Mail::Message::Head::Complete is realized by
  Mail::Message::Head::Delayed
  Mail::Message::Head::Subset
```

SYNOPSIS

```
my $head = Mail::Message::Head::Complete->new;
See Mail::Message::Head
```

DESCRIPTION

E−mail's message can be in various states: unread, partially read, and fully read. The class stores a message of which all header lines are known for sure.

Extends ''DESCRIPTION'' in Mail::Message::Head.

OVERLOADED

Extends ''OVERLOADED'' in Mail::Message::Head.

overload: **""**
: Inherited, see ''OVERLOADED'' in Mail::Message::Head

overload: **bool**
: Inherited, see ''OVERLOADED'' in Mail::Message::Head

METHODS

Extends ''METHODS'' in Mail::Message::Head.

Constructors

Extends ''Constructors'' in Mail::Message::Head.

Mail::Message::Head::Complete−>**build**([PAIR|$field], ...)
: Undefined values are interpreted as empty field values, and therefore skipped.

`$obj`−>**clone**([@names|ARRAY|Regexps])
: Make a copy of the header, optionally limited only to the header lines specified by `@names`. See **grepNames()** on the way these fields can be used.

  example:

```
my $newhead = $head->clone('Subject', 'Received');
```

Mail::Message::Head::Complete−>**new**(%options)
: Inherited, see ''Constructors'' in Mail::Message::Head

The header

Extends ''The header'' in Mail::Message::Head.

`$obj`−>**isDelayed**()
: Inherited, see ''The header'' in Mail::Message::Head

`$obj`−>**isEmpty**()
: Inherited, see ''The header'' in Mail::Message::Head

`$obj->`**isModified**()
Inherited, see ‘‘The header’’ in Mail::Message::Head

`$obj->`**knownNames**()
Inherited, see ‘‘The header’’ in Mail::Message::Head

`$obj->`**message**([$message])
Inherited, see ‘‘The header’’ in Mail::Message::Head

`$obj->`**modified**([BOOLEAN])
Inherited, see ‘‘The header’’ in Mail::Message::Head

`$obj->`**nrLines**()
Return the number of lines needed to display this header (including the trailing newline)

`$obj->`**orderedFields**()
Inherited, see ‘‘The header’’ in Mail::Message::Head

`$obj->`**size**()
Return the number of bytes needed to display this header (including the trailing newline). On systems which use CRLF as line separator, the number of lines in the header (see **nrLines()**) must be added to find the actual size in the file.

`$obj->`**wrap**($integer)
Re-fold all fields from the header to contain at most `$integer` number of characters per line.

example: re-folding a header

```
$msg->head->wrap(78);
```

Access to the header
Extends ‘‘Access to the header’’ in Mail::Message::Head.

`$obj->`**add**(`$field` | `$line` | <$name, `$body`, [$attrs]>)
Add a field to the header. If a field is added more than once, all values are stored in the header, in the order they are added.

When a `$field` object is specified (some Mail::Message::Field instance), that will be added. Another possibility is to specify a raw header `$line`, or a header line nicely split-up in `$name` and `$body`, in which case the field constructor is called for you.

`$line` or `$body` specifications which are terminated by a new-line are considered to be correctly folded. Lines which are not terminated by a new-line will be folded when needed: new-lines will be added where required. It is strongly advised to let MailBox do the folding for you.

The return value of this method is the Mail::Message::Field object which is created (or was specified).

example:

```
my $head  = Mail::Message::Head->new;
$head->add('Subject: hi!');
$head->add(From => 'me@home');
my $field = Mail::Message::Field->new('To: you@there');
$head->add($field);
my Mail::Message::Field $s = $head->add(Sender => 'I');
```

`$obj->`**addListGroup**($object)
A *list group* is a set of header fields which contain data about a mailing list which was used to transmit the message. See Mail::Message::Head::ListGroup for details about the implementation of the `$object`.

When you have a list group prepared, you can add it later using this method. You will get your private copy of the list group data in return, because the same group can be used for multiple messages.

example: of adding a list group to a header

```
my $lg = Mail::Message::Head::ListGroup->new(...);
my $own_lg = $msg->head->addListGroup($lg);
```

$obj->**addResentGroup**($resent_group|$data)

Add a `$resent_group` (a Mail::Message::Head::ResentGroup object) to the header. If you specify `$data`, that is used to create such group first. If no `Received` line is specified, it will be created for you.

These header lines have nothing to do with the user's sense of `reply` or `forward` actions: these lines trace the e-mail transport mechanism.

example:

```
my $rg = Mail::Message::Head::ResentGroup->new(head => $head, ...);
$head->addResentGroup($rg);

my $rg = $head->addResentGroup(From => 'me');
```

$obj->**addSpamGroup**($object)

A *spam fighting group* is a set of header fields which contains data which is used to fight spam. See Mail::Message::Head::SpamGroup for details about the implementation of the `$object`.

When you have a spam group prepared, you can add it later using this method. You will get your private copy of the spam group data in return, because the same group can be used for multiple messages.

example: of adding a spam group to a header

```
my $sg = Mail::Message::Head::SpamGroup->new(...);
my $own_sg = $msg->head->addSpamGroup($sg);
```

$obj->**count**($name)

Count the number of fields with this `$name`. Most fields will return 1: only one occurrence in the header. As example, the `Received` fields are usually present more than once.

$obj->**delete**($name)

Remove the field with the specified name. If the header contained multiple lines with the same name, they will be replaced all together. This method simply call**s reset()** without replacement fields. READ THE IMPORTANT WARNING IN **removeField()**

$obj->**get**(`$name`, [$index])

Inherited, see "Access to the header" in Mail::Message::Head

$obj->**grepNames**([@names|ARRAY|Regexps])

Filter from all header fields those with names which start will any of the specified list. When no names are specified, all fields will be returned. The list is ordered as they where read from file, or added later.

The `@names` are considered regular expressions, and will all be matched case insensitive and attached to the front of the string only. You may also specify one or more prepared regexes.

example:

```
my @f  = $head->grepNames();       # same as $head->orderedFields
my @f  = $head->grepNames('X-', 'Subject', ');
my @to = $head->grepNames('To\b'); # will only select To
```

$obj->**listGroup**()

Returns a *list group* description: the set of headers which form the information about mailing list software used to transport the message. See also **addListGroup()** and **removeListGroup()**.

example: use of **listGroup()**

```
if(my $lg = $msg->head->listGroup)
{  $lg->print(\*STDERR);
   $lg->delete;
}

$msg->head->removeListGroup;
```

$obj->**names**()
Returns a full ordered list of known field names, as defined in the header. Fields which were **reset()** to be empty will still be listed here.

$obj->**print**([$fh])
Print all headers to the specified $fh, by default the selected filehandle. See **printUndisclosed()** to limit the headers to include only the public headers.

example:

```
$head->print(\*OUT);
$head->print;

my $fh = IO::File->new(...);
$head->print($fh);
```

$obj->**printSelected**($fh, <STRING|Regexp>, ...)
Like the usual **print()**, the header lines are printed to the specified $fh. In this case, however, only the fields with names as specified by STRING (case insensative) or Regexp are printed. They will stay the in-order of the source header.

example: printing only a subset of the fields

```
$head->printSelected(STDOUT, qw/Subject From To/, qr/^x\-(spam|xyz)\-/i)
```

$obj->**printUndisclosed**([$fh])
Like the usual **print()**, the header lines are printed to the specified $fh, by default the selected filehandle. In this case, however, Bcc and Resent-Bcc lines are included.

$obj->**removeContentInfo**()
Remove all body related fields from the header. The header will become partial.

$obj->**removeField**($field)
Remove the specified $field object from the header. This is useful when there are possible more than one fields with the same name, and you need to remove exactly one of them. Also have a look at **delete()**, **reset()**, and **set()**.

See also **Mail::Message::Head::Partial::removeFields()** (mind the 's' at the end of the name), which accepts a string or regular expression as argument to select the fields to be removed.

WARNING WARNING WARNING: for performance reasons, the header administration uses weak references (see Scalar::Util method **weaken()**> to figure-out which fields have been removed. A header is a hash of field for fast search and an array of weak references to remember the order of the fields, required for printing. If the field is removed from the hash, the weak-ref is set to undef and the field not printed.

However... it is easy to disturb this process. Example:
my $msg = ...; # subject ref-count = 1 + 0 = 1
$msg->head->delete('Subject'); # subject ref-count = 0 = 0: clean-up
$msg->print; # subject doesn't show: ok

But
my $msg = ...; # subject ref-count = 1 + 0 = 1
my $s = $msg->head->get('subject'); # ref-count = 1 + 1 + 0 = 2
$msg->head->delete('Subject'); # subject ref-count = 1 + 0 = 1: no clean-up

```
$msg->print;            # subject DOES show: not ok
undef $s;             # ref-count becomes 0: clean-up
$msg->print;            # subject doesn't show: ok
```

To avoid the latter situation, do not catch the field object, but only the field content. SAVE are all methods which return the text:

```
my $s = $msg->head->get('subject')->body;
my $s = $msg->head->get('subject')->unfoldedBody;
my $s = $msg->head->get('subject')->foldedBody;
my $s = $msg->head->get('subject')->foldedBody;
my $s = $msg->get('subject');
my $s = $msg->subject;
my $s = $msg->string;
```

`$obj->`**removeFields**(<STRING|Regexp>, ...)

The header object is turned into a Mail::Message::Head::Partial object which has a set of fields removed. Read about the implications and the possibilities in **Mail::Message::Head::Partial::removeFields()**.

`$obj->`**removeFieldsExcept**(<STRING|Regexp>, ...)

The header object is turned into a Mail::Message::Head::Partial object which has a set of fields removed. Read about the implications and the possibilities in **Mail::Message::Head::Partial::removeFieldsExcept()**.

`$obj->`**removeListGroup**()

Removes all fields related to mailing list administration at once. The header object is turned into a Mail::Message::Head::Partial object. Read about the implications and the possibilities in **Mail::Message::Head::Partial::removeListGroup()**.

`$obj->`**removeResentGroups**()

Removes all resent groups at once. The header object is turned into a Mail::Message::Head::Partial object. Read about the implications and the possibilities in **Mail::Message::Head::Partial::removeResentGroups()**.

`$obj->`**removeSpamGroups**()

Removes all fields which were added by various spam detection software at once. The header object is turned into a Mail::Message::Head::Partial object. Read about the implications and the possibilities in **Mail::Message::Head::Partial::removeSpamGroups()**.

`$obj->`**resentGroups**()

Returns a list of Mail::Message::Head::ResentGroup objects which each represent one intermediate point in the message's transmission in the order as they appear in the header: the most recent one first. See also **addResentGroup()** and **removeResentGroups()**.

A resent group contains a set of header fields whose names start with `Resent-*`. Before the first `Resent` line is *trace* information, which is composed of an optional `Return-Path` field and an required `Received` field.

`$obj->`**reset**($name, `@fields`)

Replace the values in the header fields named by `$name` with the values specified in the list of `@fields`. A single name can correspond to multiple repeated fields. READ THE IMPORTANT WARNING IN **removeField()**

Removing fields which are part of one of the predefined field groups is not a smart idea. You can better remove these fields as group, all together. For instance, the `'Received'` lines are part of resent groups, `'X-Spam'` is past of a spam group, and `List-Post` belongs to a list group. You can delete a whole group with **Mail::Message::Head::FieldGroup::delete()**, or with methods which are provided by Mail::Message::Head::Partial.

If FIELDS is empty, the corresponding `$name` fields will be removed. The location of removed fields in the header order will be remembered. Fields with the same name which are added later will appear

at the remembered position. This is equivalent to the **delete()** method.

example:

```
# reduce number of 'Keywords' lines to last 5)
my @keywords = $head->get('Keywords');
$head->reset('Keywords', @keywords[-5..-1]) if @keywords > 5;

# Reduce the number of Received lines to only the last added one.
my @rgs = $head->resentGroups;
shift @rgs;     # keep this one (later is added in front)
$_->delete foreach @rgs;
```

$obj->**set**($field | $line | <$name, $body, [$attrs]>)

The `set` method is similar to the **add()** method, and takes the same options. However, existing values for fields will be removed before a new value is added. READ THE IMPORTANT WARNING IN **removeField()**

$obj->**spamDetected**()

Returns whether one of the spam groups defines a report about spam. If there are not header fields in the message which relate to spam-detection software, `undef` is returned. The spamgroups which report spam are returned.

example:

```
$message->delete if $message->spamDetected;

call_spamassassin($message)
   unless defined $message->spamDetected;
```

$obj->**spamGroups**([$names])

Returns a list of Mail::Message::Head::SpamGroup objects, each collecting some lines which contain spam fighting information. When any `$names` are given, then only these groups are returned. See also **addSpamGroup()** and **removeSpamGroups()**.

In scalar context, with exactly one NAME specified, that group will be returned. With more `$names` or without `$names`, a list will be returned (which defaults to the length of the list in scalar context).

example: use of **listGroup()**

```
my @sg = $msg->head->spamGroups;
$sg[0]->print(\*STDERR);
$sg[-1]->delete;

my $sg = $msg->head->spamGroups('SpamAssassin');
```

$obj->**string**()

Returns the whole header as one scalar (in scalar context) or list of lines (list context). Triggers completion.

$obj->**study**($name, [$index])

Inherited, see ''Access to the header'' in Mail::Message::Head

About the body

Extends ''About the body'' in Mail::Message::Head.

$obj->**guessBodySize**()

Inherited, see ''About the body'' in Mail::Message::Head

$obj->**guessTimeStamp**()

Make a guess about when the message was origanally posted, based on the information found in the header's `Date` field.

For some kinds of folders, **Mail::Message::guessTimestamp()** may produce a better result, for instance by looking at the modification time of the file in which the message is stored. Also some protocols, like POP can supply that information.

`$obj->`**isMultipart**()
Inherited, see ''About the body'' in Mail::Message::Head

`$obj->`**recvstamp**()
Returns an indication about when the message was sent, but only using the `Date` field in the header as last resort: we do not trust the sender of the message to specify the correct date. See **timestamp()** when you do trust the sender.

Many spam producers fake a date, which mess up the order of receiving things. The timestamp which is produced is derived from the Received headers, if they are present, and `undef` otherwise.

The timestamp is encoded as `time` is on your system (see perldoc -f time), and as such usable for the `gmtime` and `localtime` methods.

example: of time-sorting folders with received messages

```
my $folder = $mgr->open('InBox');
my @messages = sort {$a->recvstamp <=> $b->recvstamp}
                    $folder->messages;
```

example: of time-sorting messages of mixed origin

```
my $folder = $mgr->open('MyFolder');

# Pre-calculate timestamps to be sorted (for speed)
my @stamps = map { [ ($_->timestamp || 0), $_ ] }
                     $folder->messages;

my @sorted
  = map { $_->[1] }      # get the message for the stamp
      sort {$a->[0] <=> $b->[0]}   # stamps are numerics
         @stamps;
```

`$obj->`**timestamp**()
Returns an indication about when the message was sent, with as little guessing as possible. In this case, the date as specified by the sender is trusted. See **recvstamp()** when you do not want to trust the sender.

The timestamp is encoded as `time` is on your system (see perldoc -f time), and as such usable for the `gmtime` and `localtime` methods.

Internals
Extends ''Internals'' in Mail::Message::Head.

`$obj->`**addNoRealize**($field)
Inherited, see ''Internals'' in Mail::Message::Head

`$obj->`**addOrderedFields**($fields)
Inherited, see ''Internals'' in Mail::Message::Head

`$obj->`**createFromLine**()
For some mail-folder types separate messages by a line starting with '`From `'. If a message is moved to such folder from a folder-type which does not support these separators, this method is called to produce one.

`$obj->`**createMessageId**()
Creates a message-id for this message. This method will be run when a new message is created, or a message is discovered without the message-id header field. Message-ids are required for detection of message-threads. See**messageIdPr efix()**.

`$obj->`**fileLocation**()
Inherited, see ‘‘Internals’’ in Mail::Message::Head

`$obj->`**load**()
Inherited, see ‘‘Internals’’ in Mail::Message::Head

`$obj->`**messageIdPrefix**([$prefix, [$hostname]|CODE])
Mail::Message::Head::Complete->**messageIdPrefix**([$prefix, [$hostname]|CODE])
When options are provided, it sets a new way to create message-ids, as used by **createMessageId()**. You have two choices: either by providing a `$prefix` and optionally a `$hostname`, or a CODE reference.

The CODE reference will be called with the header as first argument. You must ensure yourself that the returned value is RFC compliant.

The `$prefix` defaults to `mailbox-$$`, the `$hostname` defaults to the return of Net::Domains's function `hostfqdn()`, or when not installed, the Sys::Hostname's function `hostname()`. Inbetween the two, a nano-second time provided by Time::HiRes is used. If that module is not available, `time` is called at the start of the program, and incremented for each newly created id.

In any case, a subroutine will be created to be used. A reference to that will be returned. When the method is called without arguments, but no subroutine is defined yet, one will be created.

example: setting a message prefix

```
$head->messageIdPrefix('prefix');
Mail::Message::Head::Complete->messageIdPrefix('prefix');
my $code = $head->messageIdPrefix('mailbox', 'nohost');

sub new_msgid()
{   my $head = shift;
    "myid-$$-${(rand 10000)}@example.com";
}

$many_msg->messageIdPrefix(\&new_msgid);
Mail::Message::Head::Complete->messageIdPrefix(&new_msgid);
```

`$obj->`**moveLocation**($distance)
Inherited, see ‘‘Internals’’ in Mail::Message::Head

`$obj->`**read**($parser)
Inherited, see ‘‘Internals’’ in Mail::Message::Head

`$obj->`**setNoRealize**($field)
Inherited, see ‘‘Internals’’ in Mail::Message::Head

Error handling
Extends ‘‘Error handling’’ in Mail::Message::Head.

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

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

`$obj->`**defaultTrace**([$level]|[$loglevel, `$tracelevel`]|[$level, `$callback`])
Mail::Message::Head::Complete->**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, [$strings]])
Mail::Message::Head::Complete−>**log**([$level, [$strings]])
Inherited, see ‘‘Error handling’’ in Mail::Reporter

`$obj`−>**logPriority**($level)
Mail::Message::Head::Complete−>**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::Message::Head.

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

DETAILS

Extends ‘‘DETAILS’’ in Mail::Message::Head.

DIAGNOSTICS

Warning: Cannot remove field `$name` from header: not found.
You ask to remove a field which is not known in the header. Using**delete()**, **r eset()**, or **set()** to do the job will not result in warnings: those methods check the existence of the field first.

Warning: Field objects have an implied name ($name)

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.

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