NAME

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

INHERITANCE

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

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

SYNOPSIS

```
my $head = Mail::Message::Head->new;
$head->add('From: me@localhost');
$head->add(From => 'me@localhost');
$head->add(Mail::Message::Field->new(From => 'me'));
my $subject = $head->get('subject');
my @rec = $head->get('received');
$head->delete('From');
```

DESCRIPTION

`Mail::Message::Head` MIME headers are part of Mail::Message messages, which are grouped in Mail::Box folders.

ATTENTION!!! most functionality about e−mail headers is described in Mail::Message::Head::Complete, which is a matured header object. Other kinds of headers will be translated to that type when time comes.

On this page, the general methods which are available on any header are described. Read about differences in the sub-class specific pages.

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

OVERLOADED

overload: **“”**

(stringifaction) The header, when used as string, will format as if **Mail::Message::Head::Complete::string()** was called, so return a nicely folder full header. An exception is made for Carp, which will get a simplified string to avoid unreadible messages from `croak` and `confess`.

example: using a header object as string

```
print $head;     # implicit stringification by print
$head->print;    # the same

print "$head";   # explicit stringication
```

overload: **bool**

When the header does not contain any lines (which is illegal, according to the RFCs), false is returned. In all other cases, a true value is produced.

METHODS

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

Constructors

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

Mail::Message::Head−>**build**([PAIR|$field]−LIST)

A fast way to construct a header with many lines. The PAIRs are `(name, content)` pairs of the header, but it is also possible to pass Mail::Message::Field objects. A Mail::Message::Head::Complete header is created by simply calling **Mail::Message::Head::Complete::build()**, and then each field is added. Double field names are

permitted.

example:

```
my $subject = Mail::Message::Field->new(Subject => 'xyz');

my $head = Mail::Message::Head->build
 ( From     => 'me@example.com'
 , To       => 'you@anywhere.aq'
 , $subject
 , Received => 'one'
 , Received => 'two'
 );

print ref $head;
 # -->  Mail::Message::Head::Complete
```

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

Create a new message header object. The object will store all the fields of a header. When you get information from the header, it will be returned to you as Mail::Message::Field objects, although the fields may be stored differently internally.

If you try to instantiate a Mail::Message::Head, you will automatically be upgraded to a Mail::Message::Head::Complete --a full head.

```
-Option    --Defined in     --Default
 field_type                   Mail::Message::Field::Fast
 log         Mail::Reporter   'WARNINGS'
 message                      undef
 modified                     <false>
 trace       Mail::Reporter   'WARNINGS'
```

field_type => CLASS

The type of objects that all the fields will have. This must be an extension of Mail::Message::Field.

log => LEVEL

message => MESSAGE

The MESSAGE where this header belongs to. Usually, this is not known at creation of the header, but sometimes it is. If not, call the **message()** method later to set it.

modified => BOOLEAN

trace => LEVEL

The header

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

Headers may only be partially read, in which case they are called delayed. This method returns true if some header information still needs to be read. Returns false if all header data has been read. Will never trigger completion.

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

Are there any fields defined in the current header? Be warned that the header will not be loaded for this: delayed headers will return true in any case.

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

Returns whether the header has been modified after being read.

example:

```
if($head->isModified) { ... }
```

`$obj->`**knownNames**()
Like **Mail::Message::Head::Complete::names()**, but only returns the known header fields, which may be less than `names` for header types which are partial. `names()` will trigger completion, where `knownNames()` does not.

`$obj->`**message**([$message])
Get (after setting) the message where this header belongs to. This does not trigger completion.

`$obj->`**modified**([BOOLEAN])
Sets the modified flag to BOOLEAN. Without value, the current setting is returned, but in that case you can better use **isModified()**. Changing this flag will not trigger header completion.

example:

```
$head->modified(1);
if($head->modified) { ... }
if($head->isModified) { ... }
```

`$obj->`**orderedFields**()
Returns the fields ordered the way they were read or added.

Access to the header

`$obj->`**get**(`$name`, [$index])
Get the data which is related to the field with the `$name`. The case of the characters in`$name` does not matter.

If there is only one data element defined for the `$name`, or if there is an `$index` specified as the second argument, only the specified element will be returned. If the field `$name` matches more than one header the return value depends on the context. In LIST context, all values will be returned in the order they are read. In SCALAR context, only the last value will be returned.

example:

```
my $head = Mail::Message::Head->new;
$head->add('Received: abc');
$head->add('Received: xyz');
$head->add('Subject: greetings');

my @rec_list   = $head->get('Received');
my $rec_scalar = $head->get('Received');
print ",@rec_list,$rec_scalar,"     # ,abc xyz, xyz,
print $head->get('Received', 0);    # abc
my @sub_list   = $head->get('Subject');
my $sub_scalar = $head->get('Subject');
print ",@sub_list,$sub_scalar,"     # ,greetings, greetings,
```

`$obj->`**study**(`$name`, [$index])
Like **get()**, but puts more effort in understanding the contents of the field. **Mail::Message::Field::study()** will be called for the field with the specified FIELDNAME, which returns Mail::Message::Field::Full objects. In scalar context only the last field with that name is returned. When an`$index` is specified, that element is returned.

About the body

`$obj->`**guessBodySize**()
Try to estimate the size of the body of this message, but without parsing the header or body. The result might be `undef` or a few percent of the real size. It may even be very far of the real value, that's why this is a guess.

`$obj->`**isMultipart**()
Returns whether the body of the related message is a multipart body. May trigger completion, when the `Content-Type` field is not defined.

Internals

`$obj`->**addNoRealize**($field)
: Add a field, like **Mail::Message::Head::Complete::add()** does, but avoid the loading of a possibly partial header. This method does not test the validity of the argument, nor flag the header as changed. This does not trigger completion.

`$obj`->**addOrderedFields**($fields)

`$obj`->**fileLocation**()
: Returns the location of the header in the file, as a pair begin and end. The begin is the first byte of the header. The end is the first byte after the header.

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

`$obj`->**moveLocation**($distance)
: Move the registration of the header in the file.

`$obj`->**read**($parser)
: Read the header information of one message into this header structure. This method is called by the folder object (some Mail::Box sub-class), which passes the `$parser` as an argument.

`$obj`->**setNoRealize**($field)
: Set a field, but avoid the loading of a possibly partial header as **set()** does. This method does not test the validity of the argument, nor flag the header as changed. This does not trigger completion.

Error handling

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

`$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->**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->**log**([$level, [$strings]])
: Inherited, see ''Error handling'' in Mail::Reporter

`$obj`->**logPriority**($level)

Mail::Message::Head->**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

Ordered header fields

Many Perl implementations make a big mistake by disturbing the order of header fields. For some fields (especially the *resent groups*, see Mail::Message::Head::ResentGroup) the order shall be maintained.

MailBox will keep the order of the fields as they were found in the source. When your add a new field, it will be added at the end. If your replace a field with a new value, it will stay in the original order.

Head class implementation

The header of a MIME message object contains a set of lines, which are called *fields* (by default represented by Mail::Message::Field objects). Dependent on the situation, the knowledge about the fields can be in one of three situations, each represented by a sub-class of this module:

- Mail::Message::Head::Complete

  In this case, it is sure that all knowledge about the header is available. When you**get()** information from the header and it is not there, it will never be there.

- Mail::Message::Head::Subset

  There is no certainty whether all header lines are known (probably not). This may be caused as result of reading a fast index file, as described in Mail::Box::MH::Index. The object is automatically transformed into a Mail::Message::Head::Complete when all header lines must be known.

- Mail::Message::Head::Partial

  A partial header is like a subset header: probably the header is incomplete. The means that you are not sure whether a **get()** for a field fails because the field is not a part of the message or that it fails because it is not yet known to the program. Where the subset header knows where to get the other fields, the partial header does not know it. It cannot hide its imperfection.

- Mail::Message::Head::Delayed

  In this case, there is no single field known. Access to this header will always trigger the loading of the full header.

Subsets of header fields

Message headers can be quite large, and therefore MailBox provides simplified access to some subsets of information. You can grab these sets of fields together, create and delete them as group.

On the moment, the following sets are defined:

- Mail::Message::Head::ResentGroup

  A *resent group* is a set of fields which is used to log one step in the transmission of the message from the original sender to the destination.

  Each step adds a set of headers to indicate when the message was received and how it was forwarded (without modification). These fields are best created using **Mail::Message::bounce()**.

- Mail::Message::Head::ListGroup

  Fields which are used to administer and log mailing list activity. Mailing list software has to play trics with the original message to be able to get the reply on that message back to the mailing list. Usually a large number of lines are added.

- Mail::Message::Head::SpamGroup

  A set of fields which contains header fields which are produced by spam detection software. You may want to remove these fields when you store a message for a longer period of time.

DIAGNOSTICS

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