NAME

MIME::Head − MIME message header (a subclass of Mail::Header)

SYNOPSIS

Before reading further, you should see MIME::Tools to make sure that you understand where this module fits into the grand scheme of things. Go on, do it now. I'll wait.

Ready? Ok...

Construction

```
### Create a new, empty header, and populate it manually:
$head = MIME::Head->new;
$head->replace('content-type', 'text/plain; charset=US-ASCII');
$head->replace('content-length', $len);

### Parse a new header from a filehandle:
$head = MIME::Head->read(\*STDIN);

### Parse a new header from a file, or a readable pipe:
$testhead = MIME::Head->from_file("/tmp/test.hdr");
$a_b_head = MIME::Head->from_file("cat a.hdr b.hdr |");
```

Output

```
### Output to filehandle:
$head->print(\*STDOUT);

### Output as string:
print STDOUT $head->as_string;
print STDOUT $head->stringify;
```

Getting field contents

```
### Is this a reply?
$is_reply = 1 if ($head->get('Subject') =~ /^Re: /);

### Get receipt information:
print "Last received from: ", $head->get('Received', 0);
@all_received = $head->get('Received');

### Print the subject, or the empty string if none:
print "Subject: ", $head->get('Subject',0);

### Too many hops?  Count 'em and see!
if ($head->count('Received') > 5) { ...

### Test whether a given field exists
warn "missing subject!" if (! $head->count('subject'));
```

Setting field contents

```
### Declare this to be an HTML header:
$head->replace('Content-type', 'text/html');
```

Manipulating field contents

```
### Get rid of internal newlines in fields:
$head->unfold;

### Decode any Q- or B-encoded-text in fields (DEPRECATED):
$head->decode;
```

Getting high-level MIME information

```
### Get/set a given MIME attribute:
unless ($charset = $head->mime_attr('content-type.charset')) {
    $head->mime_attr("content-type.charset" => "US-ASCII");
}

### The content type (e.g., "text/html"):
$mime_type     = $head->mime_type;

### The content transfer encoding (e.g., "quoted-printable"):
$mime_encoding = $head->mime_encoding;

### The recommended name when extracted:
$file_name     = $head->recommended_filename;

### The boundary text, for multipart messages:
$boundary      = $head->multipart_boundary;
```

DESCRIPTION

A class for parsing in and manipulating RFC-822 message headers, with some methods geared towards standard (and not so standard) MIME fields as specified in the various *Multipurpose Internet Mail Extensions* RFCs (starting with RFC 2045)

PUBLIC INTERFACE

Creation, input, and output

new [ARG],[OPTIONS]

Class method, inherited. Creates a new header object. Arguments are the same as those in the superclass.

from_file EXPR,OPTIONS

Class or instance method. For convenience, you can use this to parse a header object in from EXPR, which may actually be any expression that can be sent to *open()* so as to return a readable filehandle. The ''file'' will be opened, read, and then closed:

```
### Create a new header by parsing in a file:
my $head = MIME::Head->from_file("/tmp/test.hdr");
```

Since this method can function as either a class constructor *or* an instance initializer, the above is exactly equivalent to:

```
### Create a new header by parsing in a file:
my $head = MIME::Head->new->from_file("/tmp/test.hdr");
```

On success, the object will be returned; on failure, the undefined value.

The OPTIONS are the same as in *new()*, and are passed into *new()* if this is invoked as a class method.

Note: This is really just a convenience front-end onto `read()`, provided mostly for backwards-compatibility with MIME-parser 1.0.

read FILEHANDLE

Instance (or class) method. This initializes a header object by reading it in from a FILEHANDLE, until the terminating blank line is encountered. A syntax error or end-of-stream will also halt processing.

Supply this routine with a reference to a filehandle glob; e.g., `\*STDIN`:

```
### Create a new header by parsing in STDIN:
$head->read(\*STDIN);
```

On success, the self object will be returned; on failure, a false value.

Note: in the MIME world, it is perfectly legal for a header to be empty, consisting of nothing but the terminating blank line. Thus, we can't just use the formula that ''no tags equals error''.

Warning: as of the time of this writing, Mail::Header::read did not flag either syntax errors or unexpected end-of-file conditions (an EOF before the terminating blank line). MIME::ParserBase

takes this into account.

Getting/setting fields

The following are methods related to retrieving and modifying the header fields. Some are inherited from Mail::Header, but I've kept the documentation around for convenience.

add TAG,TEXT,[INDEX]

Instance method, inherited. Add a new occurrence of the field named TAG, given by TEXT:

```
### Add the trace information:
$head->add('Received',
           'from eryq.pr.mcs.net by gonzo.net with smtp');
```

Normally, the new occurrence will be *appended* to the existing occurrences. However, if the optional INDEX argument is 0, then the new occurrence will be *prepended*. If you want to be *explicit* about appending, specify an INDEX of −1.

Warning: this method always adds new occurrences; it doesn't overwrite any existing occurrences... so if you just want to *change* the value of a field (creating it if necessary), then you probably **don't** want to use this method: consider using `replace()` instead.

count TAG

Instance method, inherited. Returns the number of occurrences of a field; in a boolean context, this tells you whether a given field exists:

```
### Was a "Subject:" field given?
$subject_was_given = $head->count('subject');
```

The TAG is treated in a case-insensitive manner. This method returns some false value if the field doesn't exist, and some true value if it does.

decode [FORCE]

Instance method, DEPRECATED. Go through all the header fields, looking for RFC 1522 / RFC 2047 style ''Q'' (quoted-printable, sort of) or ''B'' (base64) encoding, and decode them in-place. Fellow Americans, you probably don't know what the hell I'm talking about. Europeans, Russians, et al, you probably do. `:-)`.

This method has been deprecated. See ''decode_headers'' in MIME::Parser for the full reasons. If you absolutely must use it and don't like the warning, then provide a FORCE:

```
"I_NEED_TO_FIX_THIS"
       Just shut up and do it.  Not recommended.
       Provided only for those who need to keep old scripts functioning.

"I_KNOW_WHAT_I_AM_DOING"
       Just shut up and do it.  Not recommended.
       Provided for those who REALLY know what they are doing.
```

What this method does. For an example, let's consider a valid email header you might get:

```
From: =?US-ASCII?Q?Keith_Moore?= <moore AT cs DOT utk DOT edu>
To: =?ISO-8859-1?Q?Keld_J=F8rn_Simonsen?= <keld AT dkuug DOT dk>
CC: =?ISO-8859-1?Q?Andr=E9_?= Pirard <PIRARD AT vm1 DOT ulg DOT ac DOT be>
Subject: =?ISO-8859-1?B?SWYgeW91IGNhbiByZWFkIHRoaXMgeW8=?=
 =?ISO-8859-2?B?dSB1bmRlcnN0YW5kIHRoZSBleGFtcGxlLg==?=
 =?US-ASCII?Q?.._cool!?=
```

That basically decodes to (sorry, I can only approximate the Latin characters with 7 bit sequences /o and 'e):

```
From: Keith Moore <moore AT cs DOT utk DOT edu>
To: Keld J/orn Simonsen <keld AT dkuug DOT dk>
CC: Andr'e  Pirard <PIRARD AT vm1 DOT ulg DOT ac DOT be>
Subject: If you can read this you understand the example... cool!
```

Note: currently, the decodings are done without regard to the character set: thus, the Q−encoding `=F8` is simply translated to the octet (hexadecimal `F8`), period. For piece-by-piece decoding of a

given field, you want the array context of `MIME::Words::decode_mimewords()`.

Warning: the CRLF+SPACE separator that splits up long encoded words into shorter sequences (see the Subject: example above) gets lost when the field is unfolded, and so decoding after unfolding causes a spurious space to be left in the field. *THEREFORE: if you're going to decode, do so BEFORE unfolding!*

This method returns the self object.

Thanks to Kent Boortz for providing the idea, and the baseline RFC−1522−decoding code.

delete TAG,[INDEX]

Instance method, inherited. Delete all occurrences of the field named TAG.

```
### Remove some MIME information:
$head->delete('MIME-Version');
$head->delete('Content-type');
```

get TAG,[INDEX]

Instance method, inherited. Get the contents of field TAG.

If a **numeric INDEX** is given, returns the occurrence at that index, or undef if not present:

```
### Print the first and last 'Received:' entries (explicitly):
print "First, or most recent: ", $head->get('received', 0);
print "Last, or least recent: ", $head->get('received',-1);
```

If **no INDEX** is given, but invoked in a **scalar** context, then INDEX simply defaults to 0:

```
### Get the first 'Received:' entry (implicitly):
my $most_recent = $head->get('received');
```

If **no INDEX** is given, and invoked in an **array** context, then *all* occurrences of the field are returned:

```
### Get all 'Received:' entries:
my @all_received = $head->get('received');
```

NOTE: The header(s) returned may end with a newline. If you don't want this, then **chomp** the return value.

get_all FIELD

Instance method. Returns the list of *all* occurrences of the field, or the empty list if the field is not present:

```
### How did it get here?
@history = $head->get_all('Received');
```

Note: I had originally experimented with having `get()` return all occurrences when invoked in an array context... but that causes a lot of accidents when you get careless and do stuff like this:

```
print "\u$field: ", $head->get($field);
```

It also made the intuitive behaviour unclear if the INDEX argument was given in an array context. So I opted for an explicit approach to asking for all occurrences.

print [OUTSTREAM]

Instance method, override. Print the header out to the given OUTSTREAM, or the currently-selected filehandle if none. The OUTSTREAM may be a filehandle, or any object that responds to a *print()* message.

The override actually lets you print to any object that responds to a *print()* method. This is vital for outputting MIME entities to scalars.

Also, it defaults to the *currently-selected* filehandle if none is given (not STDOUT!), so *please* supply a filehandle to prevent confusion.

stringify

Instance method. Return the header as a string. You can also invoke it as `as_string`.

unfold [FIELD]

Instance method, inherited. Unfold (remove newlines in) the text of all occurrences of the given FIELD. If the FIELD is omitted, *all* fields are unfolded. Returns the ‘‘self’’ object.

MIME-specific methods

All of the following methods extract information from the following fields:

```
Content-type
Content-transfer-encoding
Content-disposition
```

Be aware that they do not just return the raw contents of those fields, and in some cases they will fill in sensible (I hope) default values. Use `get()` or `mime_attr()` if you need to grab and process the raw field text.

Note: some of these methods are provided both as a convenience and for backwards-compatibility only, while others (like *recommended_filename()*) *really do have to be in MIME::Head to work properly*, since they look for their value in more than one field. However, if you know that a value is restricted to a single field, you should really use the Mail::Field interface to get it.

mime_attr ATTR,[VALUE]

A quick-and-easy interface to set/get the attributes in structured MIME fields:

```
$head->mime_attr("content-type"         => "text/html");
$head->mime_attr("content-type.charset" => "US-ASCII");
$head->mime_attr("content-type.name"    => "homepage.html");
```

This would cause the final output to look something like this:

```
Content-type: text/html; charset=US-ASCII; name="homepage.html"
```

Note that the special empty sub-field tag indicates the anonymous first sub-field.

Giving VALUE as undefined will cause the contents of the named subfield to be deleted:

```
$head->mime_attr("content-type.charset" => undef);
```

Supplying no VALUE argument just returns the attribute’s value, or undefined if it isn’t there:

```
$type = $head->mime_attr("content-type");      ### text/html
$name = $head->mime_attr("content-type.name"); ### homepage.html
```

In all cases, the new/current value is returned.

mime_encoding

Instance method. Try *real hard* to determine the content transfer encoding (e.g., `"base64"`, `"binary"`), which is returned in all-lowercase.

If no encoding could be found, the default of `"7bit"` is returned I quote from RFC 2045 section 6.1:

```
This is the default value -- that is, "Content-Transfer-Encoding: 7BIT"
is assumed if the Content-Transfer-Encoding header field is not present.
```

I do one other form of fixup: ‘‘7_bit’’, ‘‘7-bit’’, and ‘‘7 bit’’ are corrected to ‘‘7bit’’; likewise for ‘‘8bit’’.

mime_type [DEFAULT]

Instance method. Try `real hard` to determine the content type (e.g., `"text/plain"`, `"image/gif"`, `"x-weird-type"`, which is returned in all-lowercase. ‘‘Real hard’’ means that if no content type could be found, the default (usually `"text/plain"`) is returned. From RFC 2045 section 5.2:

```
Default RFC 822 messages without a MIME Content-Type header are
taken by this protocol to be plain text in the US-ASCII character
set, which can be explicitly specified as:

   Content-type: text/plain; charset=us-ascii
```

```
This default is assumed if no Content-Type header field is specified.
```

Unless this is a part of a ‘‘multipart/digest’’, in which case ‘‘message/rfc822’’ is the default. Note that you can also *set* the default, but you shouldn’t: normally only the MIME parser uses this feature.

multipart_boundary

Instance method. If this is a header for a multipart message, return the ‘‘encapsulation boundary’’ used to separate the parts. The boundary is returned exactly as given in the `Content-type:` field; that is, the leading double-hyphen (`--`) is *not* prepended.

Well, *almost* exactly... this passage from RFC 2046 dictates that we remove any trailing spaces:

```
If a boundary appears to end with white space, the white space
must be presumed to have been added by a gateway, and must be deleted.
```

Returns undef (**not** the empty string) if either the message is not multipart or if there is no specified boundary.

recommended_filename

Instance method. Return the recommended external filename. This is used when extracting the data from the MIME stream. The filename is always returned as a string in Perl’s internal format (the UTF8 flag may be on!)

Returns undef if no filename could be suggested.

NOTES

Why have separate objects for the entity, head, and body?

See the documentation for the MIME-tools distribution for the rationale behind this decision.

Why assume that MIME headers are email headers?

I quote from Achim Bohnet, who gave feedback on v.1.9 (I think he’s using the word ‘‘header’’ where I would use ‘‘field’’; e.g., to refer to ‘‘Subject:’’, ‘‘Content-type:’’, etc.):

```
There is also IMHO no requirement [for] MIME::Heads to look
like [email] headers; so to speak, the MIME::Head [simply stores]
the attributes of a complex object, e.g.:

    new MIME::Head type => "text/plain",
                   charset => ...,
                   disposition => ..., ... ;
```

I agree in principle, but (alas and dammit) RFC 2045 says otherwise. RFC 2045 [MIME] headers are a syntactic subset of RFC-822 [email] headers.

In my mind’s eye, I see an abstract class, call it MIME::Attrs, which does what Achim suggests... so you could say:

```
my $attrs = new MIME::Attrs type => "text/plain",
                            charset => ...,
                            disposition => ..., ... ;
```

We could even make it a superclass of MIME::Head: that way, MIME::Head would have to implement its interface, *and* allow itself to be initialized from a MIME::Attrs object.

However, when you read RFC 2045, you begin to see how much MIME information is organized by its presence in particular fields. I imagine that we’d begin to mirror the structure of RFC 2045 fields and subfields to such a degree that this might not give us a tremendous gain over just having MIME::Head.

Why all this ‘‘occurrence’’ and ‘‘index’’ jazz? Isn’t every field unique?

Aaaaaaaaahh....no.

Looking at a typical mail message header, it is soooooooo tempting to just store the fields as a hash of strings, one string per hash entry. Unfortunately, there’s the little matter of the `Received:` field, which (unlike `From:`, `To:`, etc.) will often have multiple occurrences; e.g.:

```
Received: from gsfc.nasa.gov by eryq.pr.mcs.net  with smtp
    (Linux Smail3.1.28.1 #5) id m0tStZ7-0007X4C;
     Thu, 21 Dec 95 16:34 CST
Received: from rhine.gsfc.nasa.gov by gsfc.nasa.gov
     (5.65/Ultrix3.0-C) id AA13596;
     Thu, 21 Dec 95 17:20:38 -0500
Received: (from eryq@localhost) by rhine.gsfc.nasa.gov
     (8.6.12/8.6.12) id RAA28069;
     Thu, 21 Dec 1995 17:27:54 -0500
Date: Thu, 21 Dec 1995 17:27:54 -0500
From: Eryq <eryq AT rhine DOT gsfc DOT nasa DOT gov>
Message-Id: <199512212227 DOT RAA28069 AT rhine DOT gsfc DOT nasa DOT gov>
To: eryq AT eryq DOT pr DOT mcs DOT net
Subject: Stuff and things
```

The `Received:` field is used for tracing message routes, and although it's not generally used for anything other than human debugging, I didn't want to inconvenience anyone who actually wanted to get at that information.

I also didn't want to make this a special case; after all, who knows what other fields could have multiple occurrences in the future? So, clearly, multiple entries had to somehow be stored multiple times... and the different occurrences had to be retrievable.

SEE ALSO

Mail::Header, Mail::Field, MIME::Words, MIME::Tools

AUTHOR

Eryq (*eryq AT zeegee DOT com*), ZeeGee Software Inc (*http://www.zeegee.com*). Dianne Skoll (dfs AT roaringpenguin DOT com) http://www.roaringpenguin.com

The more-comprehensive filename extraction is courtesy of Lee E. Brotzman, Advanced Data Solutions.