NAME

HTTP::Headers - Class encapsulating HTTP Message headers

SYNOPSIS

```
require HTTP::Headers;
$h = HTTP::Headers->new;

$h->header('Content-Type' => 'text/plain'); # set
$ct = $h->header('Content-Type'); # get
$h->remove_header('Content-Type'); # delete
```

DESCRIPTION

The `HTTP::Headers` class encapsulates HTTP-style message headers. The headers consist of attribute-value pairs also called fields, which may be repeated, and which are printed in a particular order. The field names are cases insensitive.

Instances of this class are usually created as member variables of the `HTTP::Request` and `HTTP::Response` classes, internal to the library.

The following methods are available:

`$h` = HTTP::Headers->new

Constructs a new `HTTP::Headers` object. You might pass some initial attribute-value pairs as parameters to the constructor. *E.g.*:

```
$h = HTTP::Headers->new(
Date => 'Thu, 03 Feb 1994 00:00:00 GMT',
Content_Type => 'text/html; version=3.2',
      Content_Base => 'http://www.perl.org/');
```

The constructor arguments are passed to the `header` method which is described below.

`$h`->clone

Returns a copy of this `HTTP::Headers` object.

`$h`->header(`$field`)
`$h`->header(`$field` => `$value`)
`$h`->header(`$f1` => `$v1`, `$f2` => `$v2`, ...)

Get or set the value of one or more header fields. The header field name ($field) is not case sensitive. To make the life easier for perl users who wants to avoid quoting before the => operator, you can use '_' as a replacement for '-' in header names.

The *header()* method accepts multiple ($field => `$value`) pairs, which means that you can update several fields with a single invocation.

The `$value` argument may be a plain string or a reference to an array of strings for a multi-valued field. If the `$value` is provided as `undef` then the field is removed. If the `$value` is not given, then that header field will remain unchanged.

The old value (or values) of the last of the header fields is returned. If no such field exists `undef` will be returned.

A multi-valued field will be returned as separate values in list context and will be concatenated with ", " as separator in scalar context. The HTTP spec (RFC 2616) promise that joining multiple values in this way will not change the semantic of a header field, but in practice there are cases like old-style Netscape cookies (see HTTP::Cookies) where "," is used as part of the syntax of a single field value.

Examples:

```
$header->header(MIME_Version => '1.0',
User_Agent => 'My-Web-Client/0.01');
$header->header(Accept => "text/html, text/plain, image/*");
$header->header(Accept => [qw(text/html text/plain image/*)]);
@accepts = $header->header('Accept'); # get multiple values
$accepts = $header->header('Accept'); # get values as a single string
```

$h->push_header($field => $value)
$h->push_header($f1 => $v1, $f2 => $v2, ...)
Add a new field value for the specified header field. Previous values for the same field are retained.

As for the *header()* method, the field name ($field) is not case sensitive and '_' can be used as a replacement for '-'.

The $value argument may be a scalar or a reference to a list of scalars.

```
$header->push_header(Accept => 'image/jpeg');
$header->push_header(Accept => [map "image/$_", qw(gif png tiff)]);
```

$h->init_header($field => $value)
Set the specified header to the given value, but only if no previous value for that field is set.

The header field name ($field) is not case sensitive and '_' can be used as a replacement for '-'.

The $value argument may be a scalar or a reference to a list of scalars.

$h->remove_header($field, ...)
This function removes the header fields with the specified names.

The header field names ($field) are not case sensitive and '_' can be used as a replacement for '-'.

The return value is the values of the fields removed. In scalar context the number of fields removed is returned.

Note that if you pass in multiple field names then it is generally not possible to tell which of the returned values belonged to which field.

$h->remove_content_headers
This will remove all the header fields used to describe the content of a message. All header field names prefixed with `Content-` fall into this category, as well as `Allow`, `Expires` and `Last-Modified`. RFC 2616 denotes these fields as *Entity Header Fields*.

The return value is a new HTTP::Headers object that contains the removed headers only.

$h->clear
This will remove all header fields.

$h->header_field_names
Returns the list of distinct names for the fields present in the header. The field names have case as suggested by HTTP spec, and the names are returned in the recommended "Good Practice" order.

In scalar context return the number of distinct field names.

$h->scan(&process_header_field)
Apply a subroutine to each header field in turn. The callback routine is called with two parameters; the name of the field and a single value (a string). If a header field is multi-valued, then the routine is called once for each value. The field name passed to the callback routine has case as suggested by HTTP spec, and the headers will be visited in the recommended "Good Practice" order.

Any return values of the callback routine are ignored. The loop can be broken by raising an exception (`die`), but the caller of *scan()* would have to trap the exception itself.

`$h->as_string`
`$h->as_string( $eol )`
Return the header fields as a formatted MIME header. Since it internally uses the `scan` method to build the string, the result will use case as suggested by HTTP spec, and it will follow recommended ''Good Practice'' of ordering the header fields. Long header values are not folded.

The optional `$eol` parameter specifies the line ending sequence to use. The default is ''n''. Embedded ''n'' characters in header field values will be substituted with this line ending sequence.

CONVENIENCE METHODS

The most frequently used headers can also be accessed through the following convenience methods. Most of these methods can both be used to read and to set the value of a header. The header value is set if you pass an argument to the method. The old header value is always returned. If the given header did not exist then `undef` is returned.

Methods that deal with dates/times always convert their value to system time (seconds since Jan 1, 1970) and they also expect this kind of value when the header value is set.

`$h->date`
This header represents the date and time at which the message was originated. *E.g.*:

```
$h->date(time); # set current date
```

`$h->expires`
This header gives the date and time after which the entity should be considered stale.

`$h->if_modified_since`
`$h->if_unmodified_since`
These header fields are used to make a request conditional. If the requested resource has (or has not) been modified since the time specified in this field, then the server will return a `304 Not Modified`response instead of the documen t itself.

`$h->last_modified`
This header indicates the date and time at which the resource was last modified. *E.g.*:

```
# check if document is more than 1 hour old
if (my $last_mod = $h->last_modified) {
if ($last_mod < time - 60*60) {
...
}
}
```

`$h->content_type`
The Content-Type header field indicates the media type of the message content. *E.g.*:

```
$h->content_type('text/html');
```

The value returned will be converted to lower case, and potential parameters will be chopped off and returned as a separate value if in an array context. If there is no such header field, then the empty string is returned. This makes it safe to do the following:

```
if ($h->content_type eq 'text/html') {
# we enter this place even if the real header value happens to
# be 'TEXT/HTML; version=3.0'
...
}
```

`$h->`content_type_charset
: Returns the upper-cased charset specified in the Content-Type header. In list context return the lower-cased bare content type followed by the upper-cased charset. Both values will be `undef` if not specified in the header.

`$h->`content_is_text
: Returns TRUE if the Content-Type header field indicate that the content is textual.

`$h->`content_is_html
: Returns TRUE if the Content-Type header field indicate that the content is some kind of HTML (including XHTML). This method can't be used to set Content-Type.

`$h->`content_is_xhtml
: Returns TRUE if the Content-Type header field indicate that the content is XHTML. This method can't be used to set Content-Type.

`$h->`content_is_xml
: Returns TRUE if the Content-Type header field indicate that the content is XML. This method can't be used to set Content-Type.

`$h->`content_encoding
: The Content-Encoding header field is used as a modifier to the media type. When present, its value indicates what additional encoding mechanism has been applied to the resource.

`$h->`content_length
: A decimal number indicating the size in bytes of the message content.

`$h->`content_language
: The natural language(s) of the intended audience for the message content. The value is one or more language tags as defined by RFC 1766. Eg. ''no'' for some kind of Norwegian and ''en-US'' for English the way it is written in the US.

`$h->`title
: The title of the document. In libwww-perl this header will be initialized automatically from the <TITLE>...</TITLE> element of HTML documents. *This header is no longer part of the HTTP standard.*

`$h->`user_agent
: This header field is used in request messages and contains information about the user agent originating the request. *E.g.*:

```
$h->user_agent('Mozilla/5.0 (compatible; MSIE 7.0; Windows NT 6.0)');
```

`$h->`server
: The server header field contains information about the software being used by the originating server program handling the request.

`$h->`from
: This header should contain an Internet e-mail address for the human user who controls the requesting user agent. The address should be machine-usable, as defined by RFC822. E.g.:

```
$h->from('King Kong <king@kong.com>');
```

This header is no longer part of the HTTP standard.

`$h->`referer
: Used to specify the address (URI) of the document from which the requested resource address was obtained.

The ''Free On-line Dictionary of Computing'' as this to say about the word *referer*:

```
<World-Wide Web> A misspelling of "referrer" which
somehow made it into the {HTTP} standard. A given {web
page}'s referer (sic) is the {URL} of whatever web page
contains the link that the user followed to the current
page. Most browsers pass this information as part of a
request.

(1998-10-19)
```

By popular demand `referrer` exists as an alias for this method so you can avoid this misspelling in your programs and still send the right thing on the wire.

When setting the referrer, this method removes the fragment from the given URI if it is present, as mandated by RFC2616. Note that the removal does *not* happen automatically if using the *header()*, *push_header()* or *init_header()* methods to set the referrer.

`$h->`www_authenticate
: This header must be included as part of a `401 Unauthorized` response. The field value consist of a challenge that indicates the authentication scheme and parameters applicable to the requested URI.

`$h->`proxy_authenticate
: This header must be included in a `407 Proxy Authentication Required` response.

`$h->`authorization
`$h->`proxy_authorization
: A user agent that wishes to authenticate itself with a server or a proxy, may do so by including these headers.

`$h->`authorization_basic
: This method is used to get or set an authorization header that use the ''Basic Authentication Scheme''. In array context it will return two values; the user name and the password. In scalar context it will return *''uname:password''* as a single string value.

  When used to set the header value, it expects two arguments. *E.g.*:

```
$h->authorization_basic($uname, $password);
```

  The method will croak if the `$uname` contains a colon ':'.

`$h->`proxy_authorization_basic
: Same as *authorization_basic()* but will set the ''Proxy-Authorization'' header instead.

NON-CANONICALIZED FIELD NAMES

The header field name spelling is normally canonicalized including the '_' to '-' translation. There are some application where this is not appropriate. Prefixing field names with ':' allow you to force a specific spelling. For example if you really want a header field name to show up as `foo_bar` instead of ''Foo-Bar'', you might set it like this:

```
$h->header(":foo_bar" => 1);
```

These field names are returned with the ':' intact for `$h->`header_field_names and the `$h->`scan callback, but the colons do not show in `$h->`as_string.

COPYRIGHT