NAME

HTTP::Message − HTTP style message (base class)

VERSION

version 6.18

SYNOPSIS

```
use base 'HTTP::Message';
```

DESCRIPTION

An `HTTP::Message` object contains some headers and a content body. The following methods are available: HTTP::Message 4

`$mess` = HTTP::Message
4 HTTP::Message $headers)" 4

`$mess` = HTTP::Message
`$headers`)" 4 HTTP::Message $headers, $content)" 4

`$mess` = HTTP::Message
`$headers`, `$content`)" 4 This constructs a new message object. Normally you would want construct `HTTP::Request` or `HTTP::Response` objects instead.

The optional `$header` argument should be a reference to an `HTTP::Headers` object or a plain array reference of key/value pairs. If an `HTTP::Headers` object is provided then a copy of it will be embedded into the constructed message, i.e. it will not be owned and can be modified afterwards without affecting the message.

The optional `$content` argument should be a string of bytes. HTTP::Message $str)" 4

`$mess` = HTTP::Message
`$str`)" 4 This constructs a new message object by parsing the given string.

`$mess->`headers
Returns the embedded `HTTP::Headers` object.

`$mess->`headers_as_string
`$mess->`headers_as_string(`$eol`)
Call the *as_string()* method for the headers in the message. This will be the same as

```
$mess->headers->as_string
```

but it will make your program a whole character shorter :−)

`$mess->`content
`$mess->`content(`$bytes`)
The *content()* method sets the raw content if an argument is given. If no argument is given the content is not touched. In either case the original raw content is returned.

If the `undef` argument is given, the content is reset to its default value, which is an empty string.

Note that the content should be a string of bytes. Strings in perl can contain characters outside the range of a byte. The `Encode` module can be used to turn such strings into a string of bytes.

`$mess->`add_content(`$bytes`)
The *add_content()* methods appends more data bytes to the end of the current content buffer.

`$mess->`add_content_utf8(`$string`)
The *add_content_utf8()* method appends the UTF−8 bytes representing the string to the end of the current content buffer.

`$mess->`content_ref
`$mess->`content_ref(\$bytes)
The *content_ref()* method will return a reference to content buffer string. It can be more efficient to access the content this way if the content is huge, and it can even be used for direct manipulation of the content, for instance:

```
${$res->content_ref} =~ s/\bfoo\b/bar/g;
```

This example would modify the content buffer in-place.

If an argument is passed it will setup the content to reference some external source. The *content()* and *add_content()* methods will automatically dereference scalar references passed this way. For other references *content()* will return the reference itself and *add_content()* will refuse to do anything.

`$mess->`content_charset

This returns the charset used by the content in the message. The charset is either found as the charset attribute of the `Content-Type` header or by guessing.

See <http://www.w3.org/TR/REC-html40/charset.html#spec-char-encoding> for details about how charset is determined.

`$mess->`decoded_content(`%options`)

Returns the content with any `Content-Encoding` undone and for textual content the raw content encoded to Perl's Unicode strings. If the `Content-Encoding` or `charset` of the message is unknown this method will fail by returning `undef`.

The following options can be specified.

`charset`

This override the charset parameter for text content. The value `none` can used to suppress decoding of the charset.

`default_charset`

This override the default charset guessed by *content_charset()* or if that fails "ISO-8859-1".

`alt_charset`

If decoding fails because the charset specified in the Content-Type header isn't recognized by Perl's Encode module, then try decoding using this charset instead of failing. The `alt_charset` might be specified as `none` to simply return the string without any decoding of charset as alternative.

`charset_strict`

Abort decoding if malformed characters is found in the content. By default you get the substitution character ("\x{FFFD}") in place of malformed characters.

`raise_error`

If TRUE then raise an exception if not able to decode content. Reason might be that the specified `Content-Encoding` or `charset` is not supported. If this option is FALSE, then *decoded_content()* will return `undef` on errors, but will still set $@.

`ref`

If TRUE then a reference to decoded content is returned. This might be more efficient in cases where the decoded content is identical to the raw content as no data copying is required in this case.

`$mess->`decodable

HTTP::Message::decodable()

This returns the encoding identifiers that *decoded_content()* can process. In scalar context returns a comma separated string of identifiers.

This value is suitable for initializing the `Accept-Encoding` request header field.

`$mess->`decode

This method tries to replace the content of the message with the decoded version and removes the `Content-Encoding` header. Returns TRUE if successful and FALSE if not.

If the message does not have a `Content-Encoding` header this method does nothing and returns TRUE.

Note that the content of the message is still bytes after this method has been called and you still need

to call *decoded_content()* if you want to process its content as a string.

`$mess->encode( $encoding, ... )`
: Apply the given encodings to the content of the message. Returns TRUE if successful. The "identity" (non-)encoding is always supported; other currently supported encodings, subject to availability of required additional modules, are "gzip", "deflate", "x-bzip2" and "base64".

  A successful call to this function will set the `Content-Encoding` header.

  Note that `multipart/*` or `message/*` messages can't be encoded and this method will croak if you try.

`$mess->parts`
`$mess->parts( @parts )`
`$mess->parts( \@parts )`
: Messages can be composite, i.e. contain other messages. The composite messages have a content type of `multipart/*` or `message/*`. This method give access to the contained messages.

  The argumentless form will return a list of `HTTP::Message` objects. If the content type of `$msg` is not `multipart/*` or `message/*` then this will return the empty list. In scalar context only the first object is returned. The returned message parts should be regarded as read-only (future versions of this library might make it possible to modify the parent by modifying the parts).

  If the content type of `$msg` is `message/*` then there will only be one part returned.

  If the content type is `message/http`, then the return value will be either an `HTTP::Request` or an `HTTP::Response` object.

  If a `@parts` argument is given, then the content of the message will be modified. The array reference form is provided so that an empty list can be provided. The `@parts` array should contain `HTTP::Message` objects. The `@parts` objects are owned by `$mess` after this call and should not be modified or made part of other messages.

  When updating the message with this method and the old content type of `$mess` is not `multipart/*` or `message/*`, then the content type is set to `multipart/mixed` and all other content headers are cleared.

  This method will croak if the content type is `message/*` and more than one part is provided.

`$mess->add_part( $part )`
: This will add a part to a message. The `$part` argument should be another `HTTP::Message` object. If the previous content type of `$mess` is not `multipart/*` then the old content (together with all content headers) will be made part #1 and the content type made `multipart/mixed` before the new part is added. The `$part` object is owned by `$mess` after this call and should not be modified or made part of other messages.

  There is no return value.

`$mess->clear`
: Will clear the headers and set the content to the empty string. There is no return value

`$mess->protocol`
`$mess->protocol( $proto )`
: Sets the HTTP protocol used for the message. The *protocol()* is a string like `HTTP/1.0` or `HTTP/1.1`.

`$mess->clone`
: Returns a copy of the message object.

`$mess->as_string`
`$mess->as_string( $eol )`
: Returns the message formatted as a single string.

  The optional `$eol` parameter specifies the line ending sequence to use. The default is "\n". If no

`$eol` is given then as_string will ensure that the returned string is newline terminated (even when the message content is not). No extra newline is appended if an explicit `$eol` is passed.

`$mess->dump( %opt )`
Returns the message formatted as a string. In void context print the string.

This differs from `$mess->as_string` in that it escapes the bytes of the content so that it's safe to print them and it limits how much content to print. The escapes syntax used is the same as for Perl's double quoted strings. If there is no content the string "(no content)" is shown in its place.

Options to influence the output can be passed as key/value pairs. The following options are recognized:

maxlength => `$num`
How much of the content to show. The default is 512. Set this to 0 for unlimited.

If the content is longer then the string is chopped at the limit and the string "...\n(### more bytes not shown)" appended.

no_content => `$str`
Replaces the "(no content)" marker.

prefix => `$str`
A string that will be prefixed to each line of the dump.

All methods unknown to `HTTP::Message` itself are delegated to the `HTTP::Headers` object that is part of every message. This allows convenient access to these methods. Refer to HTTP::Headers for details of these methods:

```
$mess->header( $field => $val )
$mess->push_header( $field => $val )
$mess->init_header( $field => $val )
$mess->remove_header( $field )
$mess->remove_content_headers
$mess->header_field_names
$mess->scan( \&doit )

$mess->date
$mess->expires
$mess->if_modified_since
$mess->if_unmodified_since
$mess->last_modified
$mess->content_type
$mess->content_encoding
$mess->content_length
$mess->content_language
$mess->title
$mess->user_agent
$mess->server
$mess->from
$mess->referer
$mess->www_authenticate
$mess->authorization
$mess->proxy_authorization
$mess->authorization_basic
$mess->proxy_authorization_basic
```

AUTHOR

Gisle Aas <gisle@activestate.com>

COPYRIGHT AND LICENSE