NAME

Net::HTTP – Low–level HTTP connection (client)

VERSION

version 6.19

SYNOPSIS

```
 use Net::HTTP;
 my $s = Net::HTTP->new(Host => "www.perl.com") || die $@;
 $s->write_request(GET => "/", 'User-Agent' => "Mozilla/5.0");
 my($code, $mess, %h) = $s->read_response_headers;

 while (1) {
    my $buf;
    my $n = $s->read_entity_body($buf, 1024);
    die "read failed: $!" unless defined $n;
    last unless $n;
    print $buf;
 }
```

DESCRIPTION

The `Net::HTTP` class is a low-level HTTP client. An instance of the `Net::HTTP` class represents a connection to an HTTP server. The HTTP protocol is described in RFC 2616. The `Net::HTTP` class supports `HTTP/1.0` and `HTTP/1.1`.

`Net::HTTP` is a sub-class of one of `IO::Socket::IP` (IPv6+IPv4), `IO::Socket::INET6` (IPv6+IPv4), or `IO::Socket::INET` (IPv4 only). You can mix the methods described below with reading and writing from the socket directly. This is not necessary a good idea, unless you know what you are doing.

The following methods are provided (in addition to those of `IO::Socket::INET`):

$s = Net::HTTP–>new(%options)

The `Net::HTTP` constructor method takes the same options as `IO::Socket::INET`'s as well as these:

```
  Host:            Initial host attribute value
  KeepAlive:       Initial keep_alive attribute value
  SendTE:          Initial send_te attribute_value
  HTTPVersion:     Initial http_version attribute value
  PeerHTTPVersion: Initial peer_http_version attribute value
  MaxLineLength:   Initial max_line_length attribute value
  MaxHeaderLines:  Initial max_header_lines attribute value
```

The `Host` option is also the default for `IO::Socket::INET`'s `PeerAddr`. The `PeerPort` defaults to 80 if not provided. The `PeerPort` specification can also be embedded in the `PeerAddr` by preceding it with a ":", and closing the IPv6 address on brackets "[]" if necessary: "192.0.2.1:80","[2001:db8::1]:80","any.example.com:80".

The `Listen` option provided by `IO::Socket::INET`'s constructor method is not allowed.

If unable to connect to the given HTTP server then the constructor returns `undef` and $@ contains the reason. After a successful connect, a `Net:HTTP` object is returned.

$s–>host

Get/set the default value of the `Host` header to send. The `$host` must not be set to an empty string (or `undef`) for HTTP/1.1.

$s–>keep_alive

Get/set the *keep-alive* value. If this value is TRUE then the request will be sent with headers indicating that the server should try to keep the connection open so that multiple requests can be sent.

The actual headers set will depend on the value of the `http_version` and `peer_http_version` attributes.

`$s`−>send_te
: Get/set the a value indicating if the request will be sent with a "TE" header to indicate the transfer encodings that the server can choose to use. The list of encodings announced as accepted by this client depends on availability of the following modules: `Compress::Raw::Zlib` for *deflate*, and `IO::Compress::Gunzip` for *gzip*.

`$s`−>http_version
: Get/set the HTTP version number that this client should announce. This value can only be set to "1.0" or "1.1". The default is "1.1".

`$s`−>peer_http_version
: Get/set the protocol version number of our peer. This value will initially be "1.0", but will be updated by a successful **read_response_headers()** method call.

`$s`−>max_line_length
: Get/set a limit on the length of response line and response header lines. The default is 8192. A value of 0 means no limit.

`$s`−>max_header_length
: Get/set a limit on the number of header lines that a response can have. The default is 128. A value of 0 means no limit.

`$s`−>format_request($method, `$uri`, `%headers`, [$content])
: Format a request message and return it as a string. If the headers do not include a `Host` header, then a header is inserted with the value of the `host` attribute. Headers like `Connection` and `Keep-Alive` might also be added depending on the status of the `keep_alive` attribute.

If `$content` is given (and it is non-empty), then a `Content-Length` header is automatically added unless it was already present.

`$s`−>write_request($method, `$uri`, `%headers`, [$content])
: Format and send a request message. Arguments are the same as for **format_request()**. Returns true if successful.

`$s`−>format_chunk(`$data`)
: Returns the string to be written for the given chunk of data.

`$s`−>write_chunk($data)
: Will write a new chunk of request entity body data. This method should only be used if the `Transfer-Encoding` header with a value of `chunked` was sent in the request. Note, writing zero-length data is a no-op. Use the **write_chunk_eof()** method to signal end of entity body data.

Returns true if successful.

`$s`−>format_chunk_eof(`%trailers`)
: Returns the string to be written for signaling EOF when a `Transfer-Encoding` of `chunked` is used.

`$s`−>write_chunk_eof(`%trailers`)
: Will write eof marker for chunked data and optional trailers. Note that trailers should not really be used unless is was signaled with a `Trailer` header.

Returns true if successful.

($code, `$mess`, `%headers`) = `$s`−>read_response_headers(`%opts`)
: Read response headers from server and return it. The `$code` is the 3 digit HTTP status code (see HTTP::Status) and `$mess` is the textual message that came with it. Headers are then returned as key/value pairs. Since key letter casing is not normalized and the same key can even occur multiple times, assigning these values directly to a hash is not wise. Only the `$code` is returned if this method is called in scalar context.

As a side effect this method updates the 'peer_http_version' attribute.

Options might be passed in as key/value pairs. There are currently only two options supported; `laxed` and `junk_out`.

The `laxed` option will make **read_response_headers()** more forgiving towards servers that have not learned how to speak HTTP properly. The `laxed` option is a boolean flag, and is enabled by passing in a TRUE value. The `junk_out` option can be used to capture bad header lines when `laxed` is enabled. The value should be an array reference. Bad header lines will be pushed onto the array.

The `laxed` option must be specified in order to communicate with pre–HTTP/1.0 servers that don't describe the response outcome or the data they send back with a header block. For these servers peer_http_version is set to "0.9" and this method returns (200, "Assumed OK").

The method will raise an exception (die) if the server does not speak proper HTTP or if the `max_line_length` or `max_header_length` limits are reached. If the `laxed` option is turned on and `max_line_length` and `max_header_length` checks are turned off, then no exception will be raised and this method will always return a response code.

`$n` = `$s`–>read_entity_body($buf, `$size`);

Reads chunks of the entity body content. Basically the same interface as for **read()** and **sysread()**, but the buffer offset argument is not supported yet. This method should only be called after a successful **read_response_headers()** call.

The return value will be `undef` on read errors, 0 on EOF, −1 if no data could be returned this time, otherwise the number of bytes assigned to `$buf`. The `$buf` is set to "" when the return value is −1.

You normally want to retry this call if this function returns either −1 or `undef` with `$!` as EINTR or EAGAIN (see Errno). EINTR can happen if the application catches signals and EAGAIN can happen if you made the socket non-blocking.

This method will raise exceptions (die) if the server does not speak proper HTTP. This can only happen when reading chunked data.

`%headers` = `$s`–>get_trailers

After **read_entity_body()** has returned 0 to indicate end of the entity body, you might call this method to pick up any trailers.

`$s`–>_rbuf

Get/set the read buffer content. The **read_response_headers()** and **read_entity_body()** methods use an internal buffer which they will look for data before they actually sysread more from the socket itself. If they read too much, the remaining data will be left in this buffer.

`$s`–>_rbuf_length

Returns the number of bytes in the read buffer. This should always be the same as:

```
length($s->_rbuf)
```

but might be more efficient.

SUBCLASSING

The **read_response_headers()** and **read_entity_body()** will invoke the **sysread()** method when they need more data. Subclasses might want to override this method to control how reading takes place.

The object itself is a glob. Subclasses should avoid using hash key names prefixed with `http_` and `io_`.

SEE ALSO

LWP, IO::Socket::INET, Net::HTTP::NB

AUTHOR

Gisle Aas <gisle@activestate.com>

COPYRIGHT AND LICENSE