NAME

HTTP::Tiny - A small, simple, correct HTTP/1.1 client

VERSION

version 0.043

SYNOPSIS

```
use HTTP::Tiny;

   my $response = HTTP::Tiny->new->get('http://example.com/');

die "Failed!\n" unless $response->{success};

print "$response->{status} $response->{reason}\n";

while (my ($k, $v) = each %{$response->{headers}}) {
for (ref $v eq 'ARRAY' ? @$v : $v) {
print "$k: $_\n";
}
}

print $response->{content} if length $response->{content};
```

DESCRIPTION

This is a very simple HTTP/1.1 client, designed for doing simple requests without the overhead of a large framework like LWP::UserAgent.

It is more correct and more complete than HTTP::Lite. It supports proxies and redirection. It also correctly resumes after EINTR.

If IO::Socket::IP 0.25 or later is installed, HTTP::Tiny will use it instead of IO::Socket::INET for transparent support for both IPv4 and IPv6.

Cookie support requires HTTP::CookieJar or an equivalent class.

METHODS

new

```
$http = HTTP::Tiny->new( %attributes );
```

This constructor returns a new HTTP::Tiny object. Valid attributes include:

- `agent`

  A user-agent string (defaults to 'HTTP-Tiny/$VERSION'). If `agent` ends in a space character, the default user-agent string is appended.

- `cookie_jar`

  An instance of HTTP::CookieJar or equivalent class that supports the `add` and `cookie_header` methods

- `default_headers`

  A hashref of default headers to apply to requests

- `local_address`

  The local IP address to bind to

- `keep_alive`

  Whether to reuse the last connection (if for the same scheme, host and port) (defaults to 1)

- `max_redirect`

  Maximum number of redirects allowed (defaults to 5)

- `max_size`

  Maximum response size (only when not using a data callback). If defined, responses larger than this will return an exception.

- `http_proxy`

  URL of a proxy server to use for HTTP connections (default is `$ENV{http_proxy}` if set)

- `https_proxy`

  URL of a proxy server to use for HTTPS connections (default is `$ENV{https_proxy}` if set)

- `proxy`

  URL of a generic proxy server for both HTTP and HTTPS connections (default is `$ENV{all_proxy}` if set)

- `no_proxy`

  List of domain suffixes that should not be proxied. Must be a comma-separated string or an array reference. (default is `$ENV{no_proxy}`)

- `timeout`

  Request timeout in seconds (default is 60)

- `verify_SSL`

  A boolean that indicates whether to validate the SSL certificate of an `https` connection (default is false)

- `SSL_options`

  A hashref of `SSL_*` options to pass through to IO::Socket::SSL

Exceptions from `max_size`, `timeout` or other errors will result in a pseudo-HTTP status code of 599 and a reason of "Internal Exception". The content field in the response will contain the text of the exception.

The `keep_alive` parameter enables a persistent connection, but only to a single destination scheme, host and port. Also, if any connection-relevant attributes are modified, a persistent connection will be dropped. If you want persistent connections across multiple destinations, use multiple HTTP::Tiny objects.

See "SSL SUPPORT" for more on the `verify_SSL` and `SSL_options` attributes.

get|head|put|post|delete

```
$response = $http->get($url);
$response = $http->get($url, \%options);
$response = $http->head($url);
```

These methods are shorthand for calling `request()` for the given method. The URL must have unsafe characters escaped and international domain names encoded. See `request()` for valid options and a description of the response.

The `success` field of the response will be true if the status code is 2XX.

post_form

```
$response = $http->post_form($url, $form_data);
$response = $http->post_form($url, $form_data, \%options);
```

This method executes a `POST` request and sends the key/value pairs from a form data hash or array reference to the given URL with a `content-type` of `application/x-www-form-urlencoded`. If data is provided as an array reference, the order is preserved; if provided as a hash reference, the terms are sorted on key and value for consistency. See documentation for the `www_form_urlencode` method for details on the encoding.

The URL must have unsafe characters escaped and international domain names encoded. See `request()` for valid options and a description of the response. Any `content-type` header or content in the options hashref will be ignored.

The `success` field of the response will be true if the status code is 2XX.

mirror

```
$response = $http->mirror($url, $file, \%options)
if ( $response->{success} ) {
print "$file is up to date\n";
}
```

Executes a `GET` request for the URL and saves the response body to the file name provided. The URL must have unsafe characters escaped and international domain names encoded. If the file already exists, the request will include an `If-Modified-Since` header with the modification timestamp of the file. You may specify a different `If-Modified-Since` header yourself in the `$options->{headers}` hash.

The `success` field of the response will be true if the status code is 2XX or if the status code is 304 (unmodified).

If the file was modified and the server response includes a properly formatted `Last-Modified` header, the file modification time will be updated accordingly.

request

```
$response = $http->request($method, $url);
$response = $http->request($method, $url, \%options);
```

Executes an HTTP request of the given method type ('GET', 'HEAD', 'POST', 'PUT', etc.) on the given URL. The URL must have unsafe characters escaped and international domain names encoded.

If the URL includes a "user:password" stanza, they will be used for Basic-style authorization headers. (Authorization headers will not be included in a redirected request.) For example:

```
    $http->request('GET', 'http://Aladdin:open
sesame@example.com/');
```

If the "user:password" stanza contains reserved characters, they must be percent-escaped:

```
    $http->request('GET', 'http://john%40example.com:password@example.com/');
```

A hashref of options may be appended to modify the request.

Valid options are:

- `headers`

  A hashref containing headers to include with the request. If the value for a header is an array reference, the header will be output multiple times with each value in the array. These headers over-write any default headers.

- `content`

  A scalar to include as the body of the request OR a code reference that will be called iteratively to produce the body of the request

- `trailer_callback`

  A code reference that will be called if it exists to provide a hashref of trailing headers (only used with chunked transfer-encoding)

- `data_callback`

  A code reference that will be called for each chunks of the response body received.

If the `content` option is a code reference, it will be called iteratively to provide the content body

of the request. It should return the empty string or undef when the iterator is exhausted.

If the `content` option is the empty string, no `content-type` or `content-length` headers will be generated.

If the `data_callback` option is provided, it will be called iteratively until the entire response body is received. The first argument will be a string containing a chunk of the response body, the second argument will be the in-progress response hash reference, as described below. (This allows customizing the action of the callback based on the `status` or `headers` received prior to the content body.)

The `request` method returns a hashref containing the response. The hashref will have the following keys:

- `success`

  Boolean indicating whether the operation returned a 2XX status code

- `url`

  URL that provided the response. This is the URL of the request unless there were redirections, in which case it is the last URL queried in a redirection chain

- `status`

  The HTTP status code of the response

- `reason`

  The response phrase returned by the server

- `content`

  The body of the response. If the response does not have any content or if a data callback is provided to consume the response body, this will be the empty string

- `headers`

  A hashref of header fields. All header field names will be normalized to be lower case. If a header is repeated, the value will be an arrayref; it will otherwise be a scalar string containing the value

On an exception during the execution of the request, the `status` field will contain 599, and the `content` field will contain the text of the exception.

www_form_urlencode

```
$params = $http->www_form_urlencode( $data );
   $response = );" -P $http->get(" -- http://example.com/query?$params
```

This method converts the key/value pairs from a data hash or array reference into a `x-www-form-urlencoded` string. The keys and values from the data reference will be UTF-8 encoded and escaped per RFC 3986. If a value is an array reference, the key will be repeated with each of the values of the array reference. If data is provided as a hash reference, the key/value pairs in the resulting string will be sorted by key and value for consistent ordering.

SSL SUPPORT

Direct `https` connections are supported only if IO::Socket::SSL 1.56 or greater and Net::SSLeay 1.49 or greater are installed. An exception will be thrown if a new enough versions of these modules not installed or if the SSL encryption fails. An `https` connection may be made via an `http` proxy that supports the CONNECT command (i.e. RFC 2817). You may not proxy `https` via a proxy that itself requires `https` to communicate.

SSL provides two distinct capabilities:

- Encrypted communication channel

- Verification of server identity

By default, HTTP::Tiny does not verify server identity.

Server identity verification is controversial and potentially tricky because it depends on a (usually paid) third-party Certificate Authority (CA) trust model to validate a certificate as legitimate. This discriminates against servers with self-signed certificates or certificates signed by free, community-driven CA's such as CAcert.org <http://cacert.org>.

By default, HTTP::Tiny does not make any assumptions about your trust model, threat level or risk tolerance. It just aims to give you an encrypted channel when you need one.

Setting the `verify_SSL` attribute to a true value will make HTTP::Tiny verify that an SSL connection has a valid SSL certificate corresponding to the host name of the connection and that the SSL certificate has been verified by a CA. Assuming you trust the CA, this will protect against a man-in-the-middle attack <http://en.wikipedia.org/wiki/Man-in-the-middle_attack>. If you are concerned about security, you should enable this option.

Certificate verification requires a file containing trusted CA certificates. If the Mozilla::CA module is installed, HTTP::Tiny will use the CA file included with it as a source of trusted CA's. (This means you trust Mozilla, the author of Mozilla::CA, the CPAN mirror where you got Mozilla::CA, the toolchain used to install it, and your operating system security, right?)

If that module is not available, then HTTP::Tiny will search several system-specific default locations for a CA certificate file:

- /etc/ssl/certs/ca-certificates.crt
- /etc/pki/tls/certs/ca-bundle.crt
- /etc/ssl/ca-bundle.pem

An exception will be raised if `verify_SSL` is true and no CA certificate file is available.

If you desire complete control over SSL connections, the `SSL_options` attribute lets you provide a hash reference that will be passed through to `IO::Socket::SSL::start_SSL()`, overriding any options set by HTTP::Tiny. For example, to provide your own trusted CA file:

```
SSL_options => {
SSL_ca_file => $file_path,
}
```

The `SSL_options` attribute could also be used for such things as providing a client certificate for authentication to a server or controlling the choice of cipher used for the SSL connection. See IO::Socket::SSL documentation for details.

PROXY SUPPORT

HTTP::Tiny can proxy both `http` and `https` requests. Only Basic proxy authorization is supported and it must be provided as part of the proxy URL: `http://user:pass@proxy.example.com/`.

HTTP::Tiny supports the following proxy environment variables:

- http_proxy
- https_proxy or HTTPS_PROXY
- all_proxy or ALL_PROXY

Tunnelling `https` over an `http` proxy using the CONNECT method is supported. If your proxy uses `https` itself, you can not tunnel `https` over it.

Be warned that proxying an `https` connection opens you to the risk of a man-in-the-middle attack by the proxy server.

The `no_proxy` environment variable is supported in the format of a comma-separated list of domain extensions proxy should not be used for.

Proxy arguments passed to `new` will override their corresponding environment variables.

LIMITATIONS

HTTP::Tiny is *conditionally compliant* with the HTTP/1.1 specification <http://www.w3.org/Protocols/rfc2616/rfc2616.html>. It attempts to meet all "MUST" requirements of the specification, but does not implement all "SHOULD" requirements.

Some particular limitations of note include:

- HTTP::Tiny focuses on correct transport. Users are responsible for ensuring that user-defined headers and content are compliant with the HTTP/1.1 specification.
- Users must ensure that URLs are properly escaped for unsafe characters and that international domain names are properly encoded to ASCII. See URI::Escape, URI::_punycode and Net::IDN::Encode.
- Redirection is very strict against the specification. Redirection is only automatic for response codes 301, 302 and 307 if the request method is 'GET' or 'HEAD'. Response code 303 is always converted into a 'GET' redirection, as mandated by the specification. There is no automatic support for status 305 ("Use proxy") redirections.
- There is no provision for delaying a request body using an `Expect` header. Unexpected `1XX` responses are silently ignored as per the specification.
- Only 'chunked' `Transfer-Encoding` is supported.
- There is no support for a Request-URI of '*' for the 'OPTIONS' request.

Despite the limitations listed above, HTTP::Tiny is considered feature-complete. New feature requests should be directed to HTTP::Tiny::UA.

SEE ALSO

- HTTP::Tiny::UA - Higher level UA features for HTTP::Tiny
- HTTP::Thin - HTTP::Tiny wrapper with HTTP::Request/HTTP::Response compatibility
- HTTP::Tiny::Mech - Wrap WWW::Mechanize instance in HTTP::Tiny compatible interface
- IO::Socket::IP - Required for IPv6 support
- IO::Socket::SSL - Required for SSL support
- LWP::UserAgent - If HTTP::Tiny isn't enough for you, this is the "standard" way to do things
- Mozilla::CA - Required if you want to validate SSL certificates
- Net::SSLeay - Required for SSL support

SUPPORT

Bugs / Feature Requests

Please report any bugs or feature requests through the issue tracker at <https://github.com/chansen/p5-http-tiny/issues>. You will be notified automatically of any progress on your issue.

Source Code

This is open source software. The code repository is available for public review and contribution under the terms of the license.

<https://github.com/chansen/p5-http-tiny>

```
git clone https://github.com/chansen/p5-http-tiny.git
```

AUTHORS

- Christian Hansen <chansen@cpan.org>
- David Golden <dagolden@cpan.org>

CONTRIBUTORS

- Alan Gardner <gardner@pythian.com>
- Alessandro Ghedini <al3xbio@gmail.com>
- Brad Gilbert <bgills@cpan.org>
- Chris Nehren <apeiron@cpan.org>
- Chris Weyl <cweyl@alumni.drew.edu>
- Claes Jakobsson <claes@surfar.nu>
- Clinton Gormley <clint@traveljury.com>
- Craig Berry <cberry@cpan.org>
- David Mitchell <davem@iabyn.com>
- Edward Zborowski <ed@rubensteintech.com>
- Jess Robinson <castaway@desert-island.me.uk>
- Lukas Eklund <leklund@gmail.com>
- Martin J. Evans <mjegh@ntlworld.com>
- Martin-Louis Bright <mlbright@gmail.com>
- Mike Doherty <doherty@cpan.org>
- Petr PíšaX <ppisar@redhat.com>
- Serguei Trouchelle <stro@cpan.org>
- Syohei YOSHIDA <syohex@gmail.com>
- Tony Cook <tony@develop-help.com>

COPYRIGHT AND LICENSE