NAME

HTTP::Request − HTTP style request message

VERSION

version 6.36

SYNOPSIS

```
require HTTP::Request;
$request = HTTP::Request->new(GET => 'http://www.example.com/');
```

and usually used like this:

```
$ua = LWP::UserAgent->new;
$response = $ua->request($request);
```

DESCRIPTION

`HTTP::Request` is a class encapsulating HTTP style requests, consisting of a request line, some headers, and a content body. Note that the LWP library uses HTTP style requests even for non-HTTP protocols. Instances of this class are usually passed to the **request()** method of an `LWP::UserAgent` object.

`HTTP::Request` is a subclass of `HTTP::Message` and therefore inherits its methods. The following additional methods are available:

`$r` = HTTP::Request->new(`$method`, `$uri`)
`$r` = HTTP::Request->new(`$method`, `$uri`, `$header`)
`$r` = HTTP::Request->new(`$method`, `$uri`, `$header`, `$content`)

Constructs a new `HTTP::Request` object describing a request on the object `$uri` using method `$method`. The`$method` ar gument must be a string. The `$uri` argument can be either a string, or a reference to a `URI` object. The optional `$header` argument should be a reference to an `HTTP::Headers` object or a plain array reference of key/value pairs. The optional `$content` argument should be a string of bytes.

`$r` = HTTP::Request->parse(`$str`)

This constructs a new request object by parsing the given string.

`$r`->method
`$r`->method(`$val`)

This is used to get/set the method attribute. The method should be a short string like ‘‘GET’’, ‘‘HEAD’’, ‘‘PUT’’, ‘‘PATCH’’ or ‘‘POST’’.

`$r`->uri
`$r`->uri(`$val`)

This is used to get/set the uri attribute. The`$val` can be a reference to a URI object or a plain string. If a string is given, then it should be parsable as an absolute URI.

`$r`->header(`$field`)
`$r`->header(`$field` => `$value`)

This is used to get/set header values and it is inherited from `HTTP::Headers` via `HTTP::Message`. See HTTP::Headers for details and other similar methods that can be used to access the headers.

`$r`->accept_decodable

This will set the `Accept-Encoding` header to the list of encodings that **decoded_content()** can decode.

`$r`->content
`$r`->content(`$bytes`)

This is used to get/set the content and it is inherited from the `HTTP::Message` base class. See HTTP::Message for details and other methods that can be used to access the content.

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.

`$r->as_string`
`$r->as_string( $eol )`
Method returning a textual representation of the request.

EXAMPLES

Creating requests to be sent with LWP::UserAgent or others can be easy. Here are a few examples.

Simple POST

Here, we'll create a simple POST request that could be used to send JSON data to an endpoint.

```
#!/usr/bin/env perl

use strict;
use warnings;

use HTTP::Request ();
use JSON::MaybeXS qw(encode_json);

my $url = 'https://www.example.com/api/user/123';
my $header = ['Content-Type' => 'application/json; charset=UTF-8'];
my $data = {foo => 'bar', baz => 'quux'};
my $encoded_data = encode_json($data);

my $r = HTTP::Request->new('POST', $url, $header, $encoded_data);
# at this point, we could send it via LWP::UserAgent
# my $ua = LWP::UserAgent->new();
# my $res = $ua->request($r);
```

Batch POST Request

Some services, like Google, allow multiple requests to be sent in one batch. <https://developers.google.com/drive/v3/web/batch> for example. Using the `add_part` method from HTTP::Message makes this simple.

```
#!/usr/bin/env perl

use strict;
use warnings;

use HTTP::Request ();
use JSON::MaybeXS qw(encode_json);

my $auth_token = 'auth_token';
my $batch_url = 'https://www.googleapis.com/batch';
my $url = 'https://www.googleapis.com/drive/v3/files/fileId/permissions?fields
my $url_no_email = 'https://www.googleapis.com/drive/v3/files/fileId/permissi

# generate a JSON post request for one of the batch entries
my $req1 = build_json_request($url, {
    emailAddress => 'example@appsrocks.com',
    role => "writer",
    type => "user",
});

# generate a JSON post request for one of the batch entries
my $req2 = build_json_request($url_no_email, {
    domain => "appsrocks.com",
    role => "reader",
```

```
    type => "domain",
});

# generate a multipart request to send all of the other requests
my $r = HTTP::Request->new('POST', $batch_url, [
    'Accept-Encoding' => 'gzip',
    # if we don't provide a boundary here, HTTP::Message will generate
    # one for us. We could use UUID::uuid() here if we wanted.
    'Content-Type' => 'multipart/mixed; boundary=END_OF_PART'
]);

# add the two POST requests to the main request
$r->add_part($req1, $req2);
# at this point, we could send it via LWP::UserAgent
# my $ua = LWP::UserAgent->new();
# my $res = $ua->request($r);
exit();

sub build_json_request {
    my ($url, $href) = @_;
    my $header = ['Authorization' => "Bearer $auth_token", 'Content-Type' =>
    return HTTP::Request->new('POST', $url, $header, encode_json($href));
}
```

SEE ALSO

HTTP::Headers, HTTP::Message, HTTP::Request::Common, HTTP::Response

AUTHOR

Gisle Aas <gisle@activestate.com>

COPYRIGHT AND LICENSE