

**NAME**

HTTP::Request::Common – Construct common HTTP::Request objects

**VERSION**

version 6.22

**SYNOPSIS**

```
use HTTP::Request::Common;
$ua = LWP::UserAgent->new;
$ua->request(GET 'http://www.sn.no/');
$ua->request(POST 'http://somewhere/foo', [foo => bar, bar => foo]);
$ua->request(PATCH 'http://somewhere/foo', [foo => bar, bar => foo]);
$ua->request(PUT 'http://somewhere/foo', [foo => bar, bar => foo]);
$ua->request(OPTIONS 'http://somewhere/foo', [foo => bar, bar => foo]);
```

**DESCRIPTION**

This module provides functions that return newly created `HTTP::Request` objects. These functions are usually more convenient to use than the standard `HTTP::Request` constructor for the most common requests.

Note that `LWP::UserAgent` has several convenience methods, including `get`, `head`, `delete`, `post` and `put`.

The following functions are provided:

`GET $url`

`GET $url, Header => Value,...`

The `GET` function returns an `HTTP::Request` object initialized with the “GET” method and the specified URL. It is roughly equivalent to the following call

```
HTTP::Request->new (
    GET => $url,
    HTTP::Headers->new(Header => Value, ...),
)
```

but is less cluttered. What is different is that a header named `Content` will initialize the content part of the request instead of setting a header field. Note that `GET` requests should normally not have a content, so this hack makes more sense for the `PUT`, `PATCH` and `POST` functions described below.

The `get(...)` method of `LWP::UserAgent` exists as a shortcut for `$ua->request(GET ...)`.

`HEAD $url`

`HEAD $url, Header => Value,...`

Like `GET()` but the method in the request is “HEAD”.

The `head(...)` method of `LWP::UserAgent` exists as a shortcut for `$ua->request(HEAD ...)`.

`DELETE $url`

`DELETE $url, Header => Value,...`

Like `GET` but the method in the request is `DELETE`. This function is not exported by default.

`PATCH $url`

`PATCH $url, Header => Value,...`

`PATCH $url, $form_ref, Header => Value,...`

`PATCH $url, Header => Value, ..., Content => $form_ref`

`PATCH $url, Header => Value, ..., Content => $content`

The same as `POST` below, but the method in the request is `PATCH`.

```

PUT $url
PUT $url, Header => Value,...
PUT $url, $form_ref, Header => Value,...
PUT $url, Header => Value,..., Content => $form_ref
PUT $url, Header => Value,..., Content => $content
    The same as POST below, but the method in the request is PUT

```

```

OPTIONS $url
OPTIONS $url, Header => Value,...
OPTIONS $url, $form_ref, Header => Value,...
OPTIONS $url, Header => Value,..., Content => $form_ref
OPTIONS $url, Header => Value,..., Content => $content
    The same as POST below, but the method in the request is OPTIONS

```

```

POST $url
POST $url, Header => Value,...
POST $url, $form_ref, Header => Value,...
POST $url, Header => Value,..., Content => $form_ref
POST $url, Header => Value,..., Content => $content
    POST, PATCH and PUT all work with the same parameters.

```

```

    %data = ( title => 'something', body => something else' );
    $ua = LWP::UserAgent->new();
    $request = HTTP::Request::Common::POST( $url, [ %data ] );
    $response = $ua->request($request);

```

They take a second optional array or hash reference parameter `$form_ref`. The content can also be specified directly using the `Content` pseudo-header, and you may also provide the `$form_ref` this way.

The `Content` pseudo-header steals a bit of the header field namespace as there is no way to directly specify a header that is actually called “Content”. If you really need this you must update the request returned in a separate statement.

The `$form_ref` argument can be used to pass key/value pairs for the form content. By default we will initialize a request using the `application/x-www-form-urlencoded` content type. This means that you can emulate an HTML `<form>` POSTing like this:

```

POST 'http://www.perl.org/survey.cgi',
    [ name => 'Gisle Aas',
      email => 'gisle@aas.no',
      gender => 'M',
      born => '1964',
      perc => '3%',
    ];

```

This will create an `HTTP::Request` object that looks like this:

```

POST http://www.perl.org/survey.cgi
Content-Length: 66
Content-Type: application/x-www-form-urlencoded

name=Gisle%20Aas&email=gisle%40aas.no&gender=M&born=1964&perc=3%25

```

Multivalued form fields can be specified by either repeating the field name or by passing the value as an array reference.

The `POST` method also supports the `multipart/form-data` content used for *Form-based File Upload* as specified in RFC 1867. You trigger this content format by specifying a content type of `'form-data'` as one of the request headers. If one of the values in the `$form_ref` is an array

reference, then it is treated as a file part specification with the following interpretation:

```
[ $file, $filename, Header => Value... ]
[ undef, $filename, Header => Value,..., Content => $content ]
```

The first value in the array (`$file`) is the name of a file to open. This file will be read and its content placed in the request. The routine will croak if the file can't be opened. Use an `undef` as `$file` value if you want to specify the content directly with a `Content` header. The `$filename` is the filename to report in the request. If this value is undefined, then the basename of the `$file` will be used. You can specify an empty string as `$filename` if you want to suppress sending the filename when you provide a `$file` value.

If a `$file` is provided by no `Content-Type` header, then `Content-Type` and `Content-Encoding` will be filled in automatically with the values returned by `LWP::MediaTypes::guess_media_type()`

Sending my `~/profile` to the survey used as example above can be achieved by this:

```
POST 'http://www.perl.org/survey.cgi',
    Content_Type => 'form-data',
    Content      => [ name => 'Gisle Aas',
                    email => 'gisle@aa.no',
                    gender => 'M',
                    born  => '1964',
                    init  => ["$ENV{HOME}/.profile"],
                    ]
```

This will create an `HTTP::Request` object that almost looks this (the boundary and the content of your `~/profile` is likely to be different):

```
POST http://www.perl.org/survey.cgi
Content-Length: 388
Content-Type: multipart/form-data; boundary="6G+f"

--6G+f
Content-Disposition: form-data; name="name"

Gisle Aas
--6G+f
Content-Disposition: form-data; name="email"

gisle@aa.no
--6G+f
Content-Disposition: form-data; name="gender"

M
--6G+f
Content-Disposition: form-data; name="born"

1964
--6G+f
Content-Disposition: form-data; name="init"; filename=".profile"
Content-Type: text/plain

PATH=/local/perl/bin:$PATH
export PATH

--6G+f--
```

If you set the `$DYNAMIC_FILE_UPLOAD` variable (exportable) to some TRUE value, then you get back a request object with a subroutine closure as the content attribute. This subroutine will read the content of any files on demand and return it in suitable chunks. This allow you to upload arbitrary big files without using lots of memory. You can even upload infinite files like */dev/audio* if you wish; however, if the file is not a plain file, there will be no `Content-Length` header defined for the request. Not all servers (or server applications) like this. Also, if the file(s) change in size between the time the `Content-Length` is calculated and the time that the last chunk is delivered, the subroutine will `Croak`.

The `post(...)` method of `LWP::UserAgent` exists as a shortcut for `$ua->request(POST ...)`.

## SEE ALSO

`HTTP::Request`, `LWP::UserAgent`

Also, there are some examples in “EXAMPLES” in `HTTP::Request` that you might find useful. For example, batch requests are explained there.

## AUTHOR

Gisle Aas <[gisle@activestate.com](mailto:gisle@activestate.com)>

## COPYRIGHT AND LICENSE

This software is copyright (c) 1994–2017 by Gisle Aas.

This is free software; you can redistribute it and/or modify it under the same terms as the Perl 5 programming language system itself.