1   Apache2::RequestRec - Perl API for Apache request record accessors

1.1 Synopsis

```
use Apache2::RequestRec ();

# set supported by the handler HTTP methods
$allowed = $r->allowed();

# auth type
$auth_type = $r->ap_auth_type();

# QUERY_STRING
$args = $r->args();

# non-parsed-headers handler
$status = $r->assbackwards();

# how many bytes were sent
$bytes_sent = $r->bytes_sent();

# client connection record
$c = $r->connection();

# "Content-Encoding" HTTP response header
$r->content_encoding("gzip");

# the languages of the content
$languages = $r->content_languages();

# "Content-Encoding" HTTP response header
$r->content_type('text/plain');

# special response headers table
$err_headers_out = $r->err_headers_out();

# request mapped filename
$filename = $r->filename();

# request finfo
$finfo = $r->finfo();

# 'SetHandler perl-script' equivalent
$r->handler('perl-script');


# was it a HEAD request?
$status = $r->header_only();

# request input headers table
$headers_in = $r->headers_in();

# request output headers table
$headers_out = $r->headers_out();

# hostname
$hostname = $r->hostname();
```

```
# input filters stack
$input_filters = $r->input_filters();

# get the main request obj in a sub-request
$main_r = $r->main();

# what's the current request (GET/POST/etc)?
$method = $r->method();

# what's the current method number?
$methnum = $r->method_number();

# current resource last modified time
$mtime = $r->mtime();

# next request object (in redirect)
$next_r = $r->next();

# there is no local copy
$r->no_local_copy();

# Apache ascii notes table
$notes = $r->notes();

# output filters stack
$output_filters = $r->output_filters();

# PATH_INFO
$path_info = $r->path_info();

# used in configuration directives modules
$per_dir_config = $r->per_dir_config();

# pool with life span of the current request
$p = $r->pool();

# previous request object in the internal redirect
$prev_r = $r->prev();

# connection level input filters stack
$proto_input_filters = $r->proto_input_filters();

# HTTP protocol version number
$proto_num = $r->proto_num();

# connection level output filters stack
$proto_output_filters = $r->proto_output_filters();

# the protocol, the client speaks: "HTTP/1.0", "HTTP/1.1", etc.
$protocol = $r->protocol();

# is it a proxy request
$status = $r->proxyreq($val);

# Time when the request started
$request_time = $r->request_time();
```

```
# server object
$s = $r->server();

# response status
$status = $r->status();

# response status line
$status_line = $r->status_line();

# manipulate %ENV of the subprocess

$r->subprocess_env;
$r->subprocess_env($key => $val);

# first HTTP request header
$request = $r->the_request();

# the URI without any parsing performed
$unparsed_uri = $r->unparsed_uri();

# The path portion of the URI
$uri = $r->uri();

# auth username
$user = $r->user();
```

1.2 Description

`Apache2::RequestRec` provides the Perl API for Apache request_rec object.

The following packages extend the `Apache2::RequestRec` functionality: `Apache2::Access`, `Apache2::Log`, `Apache2::RequestIO`, `Apache2::RequestUtil`, `Apache2::Response`, `Apache2::SubRequest` and `Apache2::URI`.

1.3 API

`Apache2::RequestRec` provides the following functions and/or methods:

1.3.1 `allowed`

Get/set the allowed methods bitmask.

```
$allowed      = $r->allowed();
$prev_allowed = $r->allowed($new_allowed);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$new_allowed` (bitmask)**

Set the bitvector.

- **ret: `$allowed` (bitmask)**

  returns `$allowed`, which is a bitvector of the allowed methods.

  If the `$new_allowed` argument is passed, the value before the change is returned.

- **since: 2.0.00**

A handler must ensure that the request method is one that it is capable of handling. Generally modules should `Apache2::DECLINE` any request methods they do not handle. Prior to aborting the handler like this the handler should set `$r->allowed` to the list of methods that it is willing to handle. This bitvector is used to construct the `"Allow:"` header required for `OPTIONS` requests, and `Apache2::Const::HTTP_METHOD_NOT_ALLOWED` (405) and `Apache2::Const::HTTP_NOT_IMPLEMENTED` (501) status codes.

Since the default Apache handler deals with the `OPTIONS` method, all response handlers can usually decline to deal with `OPTIONS`. For example if the response handler handles only `GET` and `POST` methods, and not `OPTIONS`, it may want to say:

```
use Apache2::Const -compile => qw(OK DECLINED M_GET M_POST M_OPTIONS);
if ($r->method_number == Apache2::Const::M_OPTIONS) {
    $r->allowed($r->allowed | (1<<Apache2::Const::M_GET) | (1<<Apache2::Const::M_POST));
    return Apache2::Const::DECLINED;
}
```

`TRACE` is always allowed, modules don't need to set it explicitly.

Since the default_handler will always handle a `GET`, a module which does *not* implement `GET` should probably return `Apache2::Const::HTTP_METHOD_NOT_ALLOWED`. Unfortunately this means that a script `GET` handler can't be installed by mod_actions.

For example, if the module can handle only POST method it could start with:

```
use Apache2::Const -compile => qw(M_POST HTTP_METHOD_NOT_ALLOWED);
unless ($r->method_number == Apache2::Const::M_POST) {
    $r->allowed($r->allowed | (1<<Apache2::Const::M_POST));
    return Apache2::Const::HTTP_METHOD_NOT_ALLOWED;
}
```

1.3.2 `ap_auth_type`

If an authentication check was made, get or set the *ap_auth_type* slot in the request record

```
$auth_type = $r->ap_auth_type();
$r->ap_auth_type($newval);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $newval (string)**

    If this argument is passed then a new auth type is assigned. For example:

    ```
    $r->auth_type('Basic');
    ```

- **ret: $auth_type (string)**

    If $newval is passed, nothing is returned. Otherwise the current auth type is returned.

- **since: 2.0.00**

ap_auth_type holds the authentication type that has been negotiated between the client and server during the actual request. Generally, *ap_auth_type* is populated automatically when you call $r->get_basic_auth_pw so you don't really need to worry too much about it, but if you want to roll your own authentication mechanism then you will have to populate *ap_auth_type* yourself.

Note that $r->ap_auth_type was $r->connection->auth_type in the mod_perl 1.0 API.

1.3.3 args

Get/set the request QUERY string

```
$args      = $r->args();
$prev_args = $r->args($new_args);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_args (string)**

    Optionally set the new QUERY string

- **ret: $args (string)**

    The current QUERY string

    If $new_args was passed, returns the value before the change.

- **since: 2.0.00**

1.3.4 assbackwards

When set to a true value, Apache won't send any HTTP response headers allowing you to send any headers.

```
$status      = $r->assbackwards();
$prev_status = $r->assbackwards($newval);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $newval (integer)**

    assign a new state.

- **ret: $status (integer)**

    current state.

- **since: 2.0.00**

If you send your own set of headers, which includes the `Keep-Alive` HTTP response header, you must make sure to increment the number of requests served over this connection (which is normally done by the core connection output filter `ap_http_header_filter`, but skipped when `assbackwards` is enabled).

```
$r->connection->keepalives($r->connection->keepalives + 1);
```

otherwise code relying on the value of `$r->connection->keepalives` may malfunction. For example, this counter is used to tell when a new request is coming in over the same connection to a filter that wants to parse only HTTP headers (like `Apache2::Filter::HTTPHeadersFixup`). Of course you will need to set `$r->connection->keepalive(1)`) as well.

1.3.5 `bytes_sent`

The number of bytes sent to the client, handy for logging, etc.

```
$bytes_sent = $r->bytes_sent();
```

- **obj: $r (Apache2::RequestRec object)**
- **ret: $bytes_sent (integer)**
- **since: 2.0.00**

Though as of this writing in Apache 2.0 it doesn't really do what it did in Apache 1.3. It's just set to the size of the response body. The issue is that buckets from one request may get buffered and not sent during the lifetime of the request, so it's not easy to give a truly accurate count of "bytes sent to the network for this response".

1.3.6 `connection`

Get the client connection record

```
$c = $r->connection();
```

- **obj: $r (Apache2::RequestRec object)**
- **ret: $c (Apache2::Connection object)**
- **since: 2.0.00**

1.3.7 `content_encoding`

Get/set content encoding (the "Content-Encoding" HTTP header). Content encodings are string like *"gzip"* or *"compress"*.

```
$ce      = $r->content_encoding();
$prev_ce = $r->content_encoding($new_ce);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_ce (string)**

   If passed, sets the content encoding to a new value. It must be a lowercased string.

- **ret: $ce (string)**

   The current content encoding.

   If $new_ce is passed, then the previous value is returned.

- **since: 2.0.00**

For example, here is how to send a gzip'ed response:

```
require Compress::Zlib;
$r->content_type("text/plain");
$r->content_encoding("gzip");
$r->print(Compress::Zlib::memGzip("some text to be gzipped));
```

1.3.8 `content_languages`

Get/set content languages (the `"Content-Language"` HTTP header). Content languages are string like *"en"* or *"fr"*.

```
$languages = $r->content_languages();
$prev_lang = $r->content_languages($nev_lang);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_lang (ARRAY ref)**

   If passed, sets the content languages to new values. It must be an ARRAY reference of language names, like *"en"* or *"fr"*

- **ret: $languages (ARRAY ref)**

   The current list of content languages, as an ARRAY reference.

   If $new_lang is passed, then the previous value is returned.

- **since: 2.0.00**

1.3.9 *content_type*

Get/set the HTTP response *Content-type* header value.

```
my $content_type      = $r->content_type();
my $prev_content_type = $r->content_type($new_content_type);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_content_type (MIME type string)**

  Assign a new HTTP response content-type. It will affect the response only if HTTP headers weren't sent yet.

- **ret: $content_type**

  The current content-type value.

  If $new_content_type was passed, the previous value is returned instead.

- **since: 2.0.00**

For example, set the Content-type header to *text/plain*.

```
$r->content_type('text/plain');
```

If you set this header via the headers_out table directly, it will be ignored by Apache. So do not do that.

1.3.10 *err_headers_out*

Get/set MIME response headers, printed even on errors and persist across internal redirects.

```
$err_headers_out = $r->err_headers_out();
```

- **obj: $r (Apache2::RequestRec object)**
- **ret: $err_headers_out (APR::Table object)**
- **since: 2.0.00**

The difference between headers_out and err_headers_out, is that the latter are printed even on error, and persist across internal redirects (so the headers printed for ErrorDocument handlers will have them).

For example, if a handler wants to return a 404 response, but nevertheless to set a cookie, it has to be:

```
$r->err_headers_out->add('Set-Cookie' => $cookie);
return Apache2::Const::NOT_FOUND;
```

If the handler does:

```
$r->headers_out->add('Set-Cookie' => $cookie);
return Apache2::Const::NOT_FOUND;
```

the `Set-Cookie` header won't be sent.

1.3.11 `filename`

Get/set the filename on disk corresponding to this response (the result of the *URI --> filename* translation).

```
$filename       = $r->filename();
$prev_filename = $r->filename($new_filename);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_filename (string)**

    new value

- **ret: $filename (string)**

    the current filename, or the previous value if the optional `$new_filename` argument was passed

- **since: 2.0.00**

Note that if you change the filename after the `PerlMapToStorageHandler` phase was run and expect Apache to serve it, you need to update its `stat` record, like so:

```
use Apache2::RequestRec ();
use APR::Finfo ();
use APR::Const -compile => qw(FINFO_NORM);
$r->filename($newfile);
$r->finfo(APR::Finfo::stat($newfile, APR::Const::FINFO_NORM, $r->pool));
```

if you don't, Apache will still try to use the previously cached information about the previously set value of the filename.

1.3.12 `finfo`

Get and set the *finfo* request record member:

```
$finfo = $r->finfo();
$r->finfo($finfo);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $finfo (APR::Finfo object)**
- **ret: $finfo (APR::Finfo object)**

Always returns the current object.

Due to the internal Apache implementation it's not possible to have two different objects originating from $r->finfo at the same time. Whenever $r->finfo is updated all objects will be updated too to the latest value.

- **since: 2.0.00**

Most of the time, this method is used to get the finfo member. The only reason you may want to set it is you need to use it before the Apache's default map_to_storage phase is called.

Examples:

- What Apache thinks is the current request filename (post the PerlMapToStorageHandler phase):

  ```
  use Apache2::RequestRec ();
  use APR::Finfo ();
  print $r->finfo->fname;
  ```

- Populate the finfo member (normally, before the PerlMapToStorageHandler phase):

  ```
  use APR::Finfo ();
  use APR::Const -compile => qw(FINFO_NORM);

  my $finfo = APR::Finfo::stat(__FILE__, APR::Const::FINFO_NORM, $r->pool);
  $r->finfo($finfo);
  ```

1.3.13 *handler*

Get/set the equivalent of the SetHandler directive.

```
$handler      = $r->handler();
$prev_handler = $r->handler($new_handler);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_handler (string)**

  the new handler.

- **ret: $handler (string)**

  the current handler.

  If $new_handler is passed, the previous value is returned.

- **since: 2.0.00**

1.3.14 `header_only`

Did the client has asked for headers only? e.g. if the request method was **HEAD**.

```
$status = $r->header_only();
```

- **obj: $r (Apache2::RequestRec object)**
- **ret: $status (boolean)**

    Returns true if the client is asking for headers only, false otherwise

- **since: 2.0.00**

1.3.15 `headers_in`

Get/set the request MIME headers:

```
$headers_in = $r->headers_in();
```

- **obj: $r (Apache2::RequestRec object)**
- **ret: $headers_in (APR::Table object)**
- **since: 2.0.00**

This table is available starting from the `PerlHeaderParserHandler` phase.

For example you can use it to retrieve the cookie value sent by the client, in the `Cookie:` header:

```
my $cookie = $r->headers_in->{Cookie} || '';
```

1.3.16 `headers_out`

Get/set MIME response headers, printed only on 2xx responses.

```
$headers_out = $r->headers_out();
```

- **obj: $r (Apache2::RequestRec object)**
- **ret: $headers_out (APR::Table object)**
- **since: 2.0.00**

See also `err_headers_out`, which allows to set headers for non-2xx responses and persist across internal redirects.

1.3.17 `hostname`

Host, as set by full URI or Host:

```
$hostname = $r->hostname();
$prev_hostname = $r->hostname($new_hostname);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_hostname (string)**

  new value

- **ret: $hostname (string)**

  the current hostname, or the previous value if the optional $new_hostname argument was passed

- **since: 2.0.00**

1.3.18 `input_filters`

Get/set the first filter in a linked list of request level input filters:

```
$input_filters     = $r->input_filters();
$prev_input_filters = $r->input_filters($new_input_filters);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_input_filters**

  Set a new value

- **ret: $input_filters (Apache2::Filter object)**

  The first filter in the request level input filters chain.

  If $new_input_filters was passed, returns the previous value.

- **since: 2.0.00**

For example instead of using $r->read() to read the POST data, one could use an explicit walk through incoming bucket brigades to get that data. The following function read_post() does just that (in fact that's what $r->read() does behind the scenes):

```
use APR::Brigade ();
use APR::Bucket ();
use Apache2::Filter ();

use Apache2::Const -compile => qw(MODE_READBYTES);
use APR::Const     -compile => qw(SUCCESS BLOCK_READ);

use constant IOBUFSIZE => 8192;

sub read_post {
    my $r = shift;

    my $bb = APR::Brigade->new($r->pool,
                               $r->connection->bucket_alloc);
```

```
      my $data = '';
      my $seen_eos = 0;
      do {
          $r->input_filters->get_brigade($bb, Apache2::Const::MODE_READBYTES,
                                    APR::Const::BLOCK_READ, IOBUFSIZE);

          for (my $b = $bb->first; $b; $b = $bb->next($b)) {
              if ($b->is_eos) {
                  $seen_eos++;
                  last;
              }

              if ($b->read(my $buf)) {
                  $data .= $buf;
              }

              $b->remove; # optimization to reuse memory
          }

      } while (!$seen_eos);

      $bb->destroy;

      return $data;
  }
```

As you can see `$r->input_filters` gives us a pointer to the last of the top of the incoming filters stack.

1.3.19 `main`

Get the main request record

```
$main_r = $r->main();
```

- **obj: $r (Apache2::RequestRec object)**
- **ret: $main_r (Apache2::RequestRec object)**

    If the current request is a sub-request, this method returns a blessed reference to the main request structure. If the current request is the main request, then this method returns `undef`.

    To figure out whether you are inside a main request or a sub-request/internal redirect, use `$r->is_initial_req`.

- **since: 2.0.00**

1.3.20 `method`

Get/set the current request method (e.g. GET, HEAD, POST, etc.):

```
$method     = $r->method();
$pre_method = $r->method($new_method);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_method (string)**

  a new value

- **ret: $method (string)**

  The current method as a string

  if $new_method was passed the previous value is returned.

- **since: 2.0.00**

1.3.21 `method_number`

Get/set the HTTP method, issued by the client (Apache2::Const::M_GET, Apache2::Const::M_POST, etc.)

```
$methnum      = $r->method_number();
$prev_methnum = $r->method_number($new_methnum);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_methnum (Apache2::Const :methods constant)**

  a new value

- **ret: $methnum (Apache2::Const :methods constant)**

  The current method as a number

  if $new_methnum was passed the previous value is returned.

- **since: 2.0.00**

See the $r->allowed entry for examples.

1.3.22 `mtime`

Last modified time of the requested resource

```
$mtime      = $r->mtime();
$prev_mtime = $r->mtime($new_mtime);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_mtime (epoch seconds).**

    a new value

- **ret: $mtime (epoch seconds).**

    the current value

    if $new_mtime was passed the previous value is returned.

- **since: 2.0.00**

1.3.23 `next`

Pointer to the redirected request if this is an external redirect

```
$next_r = $r->next();
```

- **obj: $r (Apache2::RequestRec object)**
- **ret: $next_r (Apache2::RequestRec object)**

    returns a blessed reference to the next (internal) request structure or `undef` if there is no next request.

- **since: 2.0.00**

1.3.24 `no_local_copy`

There is no local copy of this response

```
$status = $r->no_local_copy();
```

- **obj: $r (Apache2::RequestRec object)**
- **ret: $status (integer)**
- **since: 2.0.00**

Used internally in certain sub-requests to prevent sending `Apache2::Const::HTTP_NOT_MODI-FIED` for a fragment or error documents. For example see the implementation in *modules/filters/mod_include.c*.

Also used internally in `$r->meets_conditions` -- if set to a true value, the conditions are always met.

1.3.25 `notes`

Get/set text notes for the duration of this request. These notes can be passed from one module to another (not only mod_perl, but modules in any other language):

```
$notes      = $r->notes();
$prev_notes = $r->notes($new_notes);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$new_notes` (`APR::Table object`)**
- **ret: `$notes` (`APR::Table object`)**

    the current notes table.

    if the `$new_notes` argument was passed, returns the previous value.

- **since: 2.0.00**

If you want to pass Perl structures, you can use `$r->pnotes`.

Also see `$c->notes`

1.3.26 `output_filters`

Get the first filter in a linked list of request level output filters:

```
$output_filters      = $r->output_filters();
$prev_output_filters = $r->output_filters($new_output_filters);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$new_output_filters`**

    Set a new value

- **ret: `$output_filters` (`Apache2::Filter object`)**

    The first filter in the request level output filters chain.

    If `$new_output_filters` was passed, returns the previous value.

- **since: 2.0.00**

For example instead of using `$r->print()` to send the response body, one could send the data directly to the first output filter. The following function `send_response_body()` does just that:

```
use APR::Brigade ();
use APR::Bucket ();
use Apache2::Filter ();

sub send_response_body {
```

```
    my ($r, $data) = @_;

    my $bb = APR::Brigade->new($r->pool,
                               $r->connection->bucket_alloc);

    my $b = APR::Bucket->new($bb->bucket_alloc, $data);
    $bb->insert_tail($b);
    $r->output_filters->fflush($bb);
    $bb->destroy;
}
```

In fact that's what $r->read() does behind the scenes. But it also knows to parse HTTP headers passed together with the data and it also implements buffering, which the above function does not.

1.3.27 `path_info`

Get/set the PATH_INFO, what is left in the path after the *URI --> filename* translation:

```
$path_info      = $r->path_info();
$prev_path_info = $r->path_info($path_info);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $path_info (string)**

    Set a new value

- **ret: $path_info (string)**

    Return the current value.

    If the optional argument $path_info is passed, the previous value is returned.

- **since: 2.0.00**

1.3.28 `per_dir_config`

Get the dir config vector:

```
$per_dir_config = $r->per_dir_config();
```

- **obj: $r (Apache2::RequestRec object)**
- **ret: $per_dir_config (Apache2::ConfVector object)**
- **since: 2.0.00**

For an indepth discussion, refer to the Apache Server Configuration Customization in Perl chapter.

1.3.29 `pool`

The pool associated with the request

```
$p = $r->pool();
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **ret: `$p` (`APR::Pool object`)**
- **since: 2.0.00**

1.3.30 `prev`

Pointer to the previous request if this is an internal redirect

```
$prev_r = $r->prev();
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **ret: `$prev_r` (`Apache2::RequestRec object`)**

  a blessed reference to the previous (internal) request structure or `undef` if there is no previous request.

- **since: 2.0.00**

1.3.31 `proto_input_filters`

Get the first filter in a linked list of protocol level input filters:

```
$proto_input_filters      = $r->proto_input_filters();
$prev_proto_input_filters = $r->proto_input_filters($new_proto_input_filters);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$new_proto_input_filters`**

  Set a new value

- **ret: `$proto_input_filters` (`Apache2::Filter object`)**

  The first filter in the protocol level input filters chain.

  If `$new_proto_input_filters` was passed, returns the previous value.

- **since: 2.0.00**

`$r->proto_input_filters` points to the same filter as `$r->connection->input_filters`.

1.3.32 `proto_num`

Get current request's HTTP protocol version number

```
$proto_num = $r->proto_num();
```

- **obj: $r (Apache2::RequestRec object)**
- **ret: $proto_num (integer)**

    current request's HTTP protocol version number, e.g.: HTTP/1.0 == 1000, HTTP/1.1 = 1001

- **since: 2.0.00**

1.3.33 `proto_output_filters`

Get the first filter in a linked list of protocol level output filters:

```
$proto_output_filters      = $r->proto_output_filters();
$prev_proto_output_filters = $r->proto_output_filters($new_proto_output_filters);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_proto_output_filters**

    Set a new value

- **ret: $proto_output_filters (Apache2::Filter object)**

    The first filter in the protocol level output filters chain.

    If $new_proto_output_filters was passed, returns the previous value.

- **since: 2.0.00**

$r->proto_output_filters points to the same filter as $r->connection->output_filters.

1.3.34 `protocol`

Get a string identifying the protocol that the client speaks.

```
$protocol = $r->protocol();
```

- **obj: $r (Apache2::RequestRec object)**
- **ret: $protocl (string)**

    Typical values are "HTTP/1.0" or "HTTP/1.1".

   If the client didn't specify the protocol version, the default is `"HTTP/0.9"`

- **since: 2.0.00**

1.3.35 `proxyreq`

Get/set the *proxyrec* request record member and optionally adjust other related fields.

```
$status = $r->proxyreq($val);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$val` (integer)**

   PROXYREQ_NONE, PROXYREQ_PROXY, PROXYREQ_REVERSE, PROXYREQ_RESPONSE

- **ret: `$status` (integer)**

   If `$val` is defined the *proxyrec* member will be set to that value and previous value will be returned.

   If `$val` is not passed, and `$r->proxyreq` is not true, and the proxy request is matching the current vhost (scheme, hostname and port), the *proxyrec* member will be set to PROX-YREQ_PROXY and that value will be returned. In addition `$r->uri` is set to `$r->unparsed_uri` and `$r->filename` is set to `"modperl-proxy:".$r->uri`. If those conditions aren't true 0 is returned.

- **since: 2.0.00**

For example to turn a normal request into a proxy request to be handled on the same server in the `Perl-TransHandler` phase run:

```
my $real_url = $r->unparsed_uri;
$r->proxyreq(Apache2::Const::PROXYREQ_PROXY);
$r->uri($real_url);
$r->filename("proxy:$real_url");
$r->handler('proxy-server');
```

Also remember that if you want to turn a proxy request into a non-proxy request, it's not enough to call:

```
$r->proxyreq(Apache2::Const::PROXYREQ_NONE);
```

You need to adjust `$r->uri` and `$r->filename` as well if you run that code in `PerlPostRead-RequestHandler` phase, since if you don't -- mod_proxy's own post_read_request handler will override your settings (as it will run after the mod_perl handler).

And you may also want to add

```
$r->set_handlers(PerlResponseHandler => []);
```

so that any response handlers which match apache directives will not run in addition to the mod_proxy content handler.

1.3.36 request_time

Time when the request started

```
$request_time = $r->request_time();
```

- **obj: $r (Apache2::RequestRec object)**
- **ret: $request_time (epoch seconds).**
- **since: 2.0.00**

1.3.37 server

Get the `Apache2::Server` object for the server the request `$r` is running under.

```
$s = $r->server();
```

- **obj: $r (Apache2::RequestRec object)**
- **ret: $s (Apache2::ServerRec object)**
- **since: 2.0.00**

1.3.38 status

Get/set the reply status for the client request.

```
$status      = $r->status();
$prev_status = $r->status($new_status);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_status (integer)**

  If `$new_status` is passed the new status is assigned.

  Normally you would use some `Apache2::Const constant`, e.g.
  `Apache2::Const::REDIRECT`.

- **ret: $newval (integer)**

  The current value.

  If `$new_status` is passed the old value is returned.

- **since: 2.0.00**

Usually you will set this value indirectly by returning the status code as the handler's function result. However, there are rare instances when you want to trick Apache into thinking that the module returned an `Apache2::Const::OK` status code, but actually send the browser a non-OK status. This may come handy when implementing an HTTP proxy handler. The proxy handler needs to send to the client, whatever status code the proxied server has returned, while returning `Apache2::Const::OK` to Apache. e.g.:

```
$r->status($some_code);
return Apache2::Const::OK
```

See also `$r->status_line`, which. if set, overrides `$r->status`.

1.3.39 *status_line*

Get/set the response status line. The status line is a string like "200 Document follows" and it will take precedence over the value specified using the `$r->status()` described above.

```
$status_line      = $r->status_line();
$prev_status_line = $r->status_line($new_status_line);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$new_status_line` (string)**
- **ret: `$status_line` (string)**
- **since: 2.0.00**

When discussing `$r->status` we have mentioned that sometimes a handler runs to a successful completion, but may need to return a different code, which is the case with the proxy server. Assuming that the proxy handler forwards to the client whatever response the proxied server has sent, it'll usually use `status_line()`, like so:

```
$r->status_line($response->code() . ' ' . $response->message());
return Apache2::Const::OK;
```

In this example `$response` could be for example an `HTTP::Response` object, if `LWP::UserAgent` was used to implement the proxy.

This method is also handy when you extend the HTTP protocol and add new response codes. For example you could invent a new error code and tell Apache to use that in the response like so:

```
$r->status_line("499 We have been FooBared");
return Apache2::Const::OK;
```

Here `499` is the new response code, and *We have been FooBared* is the custom response message.

1.3.40 `subprocess_env`

Get/set the Apache `subprocess_env` table, or optionally set the value of a named entry.

```
              $r->subprocess_env;
$env_table = $r->subprocess_env;

      $r->subprocess_env($key => $val);
$val = $r->subprocess_env($key);
```

- **obj: `$r` (`Apache2::RequestRec object`)**
- **opt arg1: `$key` (string)**
- **opt arg2: `$val` (string)**
- **ret: `...`**
- **since: 2.0.00**

When called in VOID context with no arguments, it populate `%ENV` with special variables (e.g. `$ENV{QUERY_STRING}`) like mod_cgi does.

When called in a non-VOID context with no arguments, it returns an `APR::Table object`.

When the `$key` argument (string) is passed, it returns the corresponding value (if such exists, or `undef`. The following two lines are equivalent:

```
$val = $r->subprocess_env($key);
$val = $r->subprocess_env->get($key);
```

When the `$key` and the `$val` arguments (strings) are passed, the value is set. The following two lines are equivalent:

```
$r->subprocess_env($key => $val);
$r->subprocess_env->set($key => $val);
```

The `subprocess_env table` is used by `Apache2::SubProcess`, to pass environment variables to externally spawned processes. It's also used by various Apache modules, and you should use this table to pass the environment variables. For example if in `PerlHeaderParserHandler` you do:

```
$r->subprocess_env(MyLanguage => "de");
```

you can then deploy `mod_include` and write in *.shtml* document:

```
<!--#if expr="$MyLanguage = en" -->
English
<!--#elif expr="$MyLanguage = de" -->
Deutsch
<!--#else -->
Sorry
<!--#endif -->
```

1.3.41 *the_request*

First HTTP request header

```
$request = $r->the_request();
$old_request = $r->uri($new_request);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_request (string)**
- **ret: $request (string)**

For example:

```
GET /foo/bar/my_path_info?args=3 HTTP/1.0
```

- **since: 2.0.00**

1.3.42 *unparsed_uri*

The URI without any parsing performed

```
$unparsed_uri = $r->unparsed_uri();
```

- **obj: $r (Apache2::RequestRec object)**
- **ret: $unparsed_uri (string)**
- **since: 2.0.00**

If for example the request was:

```
GET /foo/bar/my_path_info?args=3 HTTP/1.0
```

`$r->uri` returns:

```
/foo/bar/my_path_info
```

whereas `$r->unparsed_uri` returns:

```
/foo/bar/my_path_info?args=3
```

1.3.43 *uri*

The path portion of the URI

```
$uri       = $r->uri();
my $prec_uri = $r->uri($new_uri);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $new_uri (string)**
- **ret: $uri (string)**

- **since: 2.0.00**

See the example in the `$r->unparsed_uri` section.

1.3.44 user

Get the user name, if an authentication process was successful. Or set it.

```
$user      = $r->user();
$prev_user = $r->user($new_user);
```

- **obj: $r (`Apache2::RequestRec object`)**
- **opt arg1: $new_user (string)**

  Pass $new_user to set a new value

- **ret: $user (string)**

  The current username if an authentication process was successful.

  If $new_user was passed, the previous value is returned.

- **since: 2.0.00**

For example, let's print the username passed by the client:

```
my ($res, $sent_pw) = $r->get_basic_auth_pw;
return $res if $res != Apache2::Const::OK;
print "User: ", $r->user;
```

1.4 Unsupported API

`Apache2::RequestRec` also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

1.4.1 allowed_methods

META: Autogenerated - needs to be reviewed/completed

List of allowed methods

```
$list = $r->allowed_methods();
```

- **obj: $r (Apache2::RequestRec object)**
- **ret: $list (*Apache2::MethodList object*)**
- **since: 2.0.00**

META: Apache2::MethodList is not available at the moment

1.4.2 allowed_xmethods

META: Autogenerated - needs to be reviewed/completed

Array of extension methods

```
$array = $r->allowed_xmethods();
```

- **obj: $r (Apache2::RequestRec object)**
- **ret: $array (*APR::ArrayHeader object*)**
- **since: 2.0.00**

META: APR::ArrayHeader is not available at the moment

1.4.3 request_config

Config vector containing pointers to request's per-server config structures

```
$ret = $r->request_config($newval);
```

- **obj: $r (Apache2::RequestRec object)**
- **opt arg1: $newval (Apache2::ConfVector object)**
- **since: 2.0.00**

1.4.4 used_path_info

META: Autogenerated - needs to be reviewed/completed

Flag for the handler to accept or reject path_info on the current request. All modules should respect the AP_REQ_ACCEPT_PATH_INFO and AP_REQ_REJECT_PATH_INFO values, while AP_REQ_DEFAULT_PATH_INFO indicates they may follow existing conventions. This is set to the user's preference upon HOOK_VERY_FIRST of the fixups.

```
$ret = $r->used_path_info($newval);
```

- **obj: $r (Apache2::RequestRec object)**
- **arg1: $newval (integer)**
- **since: 2.0.00**

1.5 See Also

mod_perl 2.0 documentation.

1.6 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 2.0.

1.7 Authors

The mod_perl development team and numerous contributors.

Table of Contents: