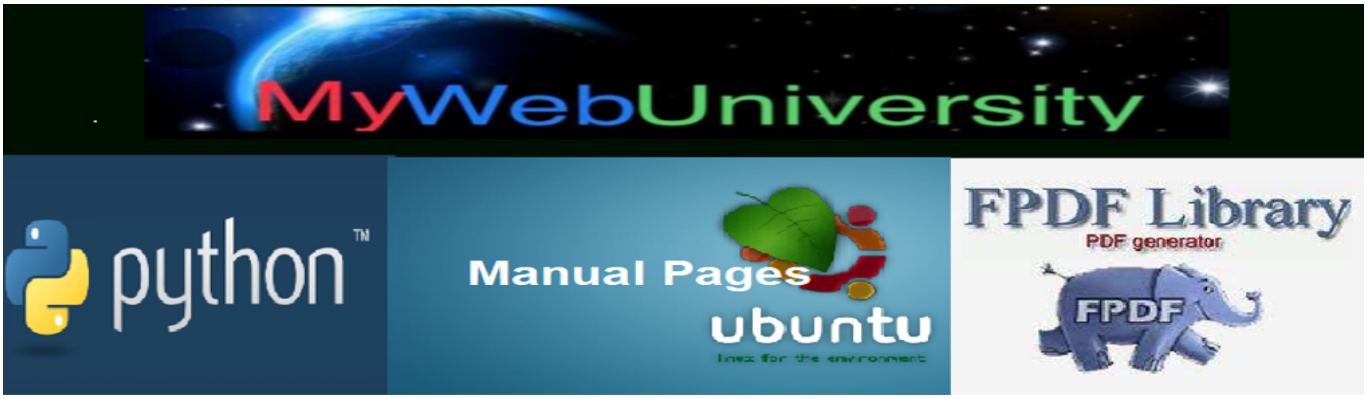




## *Rocky Enterprise Linux 9.2 Manual Pages on command 'Apache2::RequestUtil.3pm'*

***$ man Apache2::RequestUtil.3pm***

libapache2-mod-perl2-2.0.12::doUlibapache2-mod-perl2-2.0.12::docs::api::Apache2::RequestUtil(3pm)

NAME

Apache2::RequestUtil - Perl API for Apache request record utils

Synopsis

```
use Apache2::RequestUtil ();

# add httpd config dynamically

$r->add_config(['require valid-user']);

# dump the request object as a string

print $r->as_string();

# default content_type

$content_type = $r->default_type();

# get PerlSetVar/PerlAddVar values

@values = $r->dir_config->get($key);

# get server docroot

$docroot = $r->document_root();

# set server docroot

$r->document_root($new_root);

# what are the registered perl handlers for a given phase

my @handlers = @{ $r->get_handlers('PerlResponseHandler') || [] };

# push a new handler for a given phase

$r->push_handlers(PerlCleanupHandler => \&handler);

# set handlers for a given phase (resetting previous values)

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

```
# what's the request body limit
$limit = $r->get_limit_req_body();
# server and port names
$server = $r->get_server_name();
$port   = $r->get_server_port();
# what string Apache is going to send for a given status code
$status_line = Apache2::RequestUtil::get_status_line(404);
# are we in the main request?
$is_initial = $r->is_initial_req();
# directory level PerlOptions flags lookup
$r->subprocess_env unless $r->is_perl_option_enabled('SetupEnv');
# current <Location> value
$location = $r->location();
# merge a <Location> container in a request object
$r->location_merge($location);
# create a new Apache2::RequestRec object
$r = Apache2::RequestRec->new($c);
# tell the client not to cache the response
$r->no_cache($boolean);
# share perl objects by reference like $r->notes
$r->pnotes($key => [$obj1, $obj2]);
# get HTML signature
$sig = $r->psignature($prefix);
# get the global request object (requires PerlOptions +GlobalRequest)
$r = Apache2::RequestUtil->request;
# insert auth credentials into the request as if the client did that
$r->set_basic_credentials($username, $password);
# slurp the contents of $r->filename
my $content = ${ $r->slurp_filename() };
# terminate the current child after this request
$r->child_terminate();
```

Description

"Apache2::RequestUtil" provides the Apache request object utilities API.

API

"add_config"

Dynamically add Apache configuration at request processing runtime:

$r->add_config($lines);

$r->add_config($lines, $override);

$r->add_config($lines, $override, $path);

$r->add_config($lines, $override, $path, $override_opts);

Configuration directives are processed as if given in a "<Location>" block.

obj: $r ( "Apache2::RequestRec object" )

arg1: $lines (ARRAY ref)

An ARRAY reference containing configuration lines per element, without the new line terminators.

opt arg2: $override ( "Apache2::Const override constant" )

Which allow-override bits are set

Default value is: "Apache2::Const::OR_AUTHCFG"

opt arg3: $path ( string )

Set the "Apache2::CmdParms object" "path" component. This is the path of the "<Location>" block. Some directives need this, for example "ProxyPassReverse".

If an empty string is passed a "NULL" pointer is passed further at C-level. This is necessary to make something like this work:

```
$r->add_config( [
                 '<Directory />',
                 'AllowOverride Options AuthConfig',
                 '</Directory>',
                ], ~0, '' );
```

Note: "AllowOverride" is valid only in directory context.

Caution: Some directives need a non-empty path otherwise they cause segfaults. Thus, use the empty path with caution.

Default value is: "/"

opt arg4: $override_opts ( "Apache2::Const options constant" )

Apache limits the applicable directives in certain situations with "AllowOverride".

With Apache 2.2 comes the possibility to enable or disable single options, for example

AllowOverride AuthConfig Options=ExecCGI,Indexes

Internally, this directive is parsed into 2 bit fields that are represented by the $override and $override_opts parameters to "add_config". The above example is parsed into an $override with 2 bits set, one for "AuthConfig" the other for "Options" and an $override_opts with 2 bits set for ExecCGI and Indexes.

When applying other directives, for example "AuthType" or "Options" the appropriate bits in $override must be set. For the "Options" directive additionally $override_opts bits must be set.

The $override and $override_opts parameters to "add_config" are valid while applying $lines.

$override_opts is new in Apache 2.2. The mod_perl implementation for Apache 2.0 lets you pass the parameter but ignores it.

Default for $override_opts is: "Apache2::Const::OPT_UNSET" | "Apache2::Const::OPT_ALL" | "Apache2::Const::OPT_INCNOEXEC" | "Apache2::Const::OPT_SYM_OWNER" | "Apache2::Const::OPT_MULTI"

That means, all options are allowed.

ret: no return value

since: 2.0.00, $path and $override_opts since 2.0.3

See also: "$s->add_config"

For example:

```
use Apache2::RequestUtil ();
use Apache2::Access ();
$r->add_config(['require valid-user']);
# this regards the current AllowOverride setting
$r->add_config(['AuthName secret',
                'AuthType Basic',
                'Options ExecCGI'],
               $r->allow_override, $path, $r->allow_override_opts);
```

"as_string"

Dump the request object as a string

```
$dump = $r->as_string();
```

obj: $r ( "Apache2::RequestRec object" )

ret: $dump ( string )

since: 2.0.00

Dumps various request and response headers (mainly useful for debugging)

"child_terminate"

Terminate the current worker process as soon as the current request is over

```
$r->child_terminate();
```

obj: $r ( "Apache2::RequestRec object" )

ret: no return value

since: 2.0.00

This method is not supported in threaded MPMs

"default_type"

Retrieve the value of the DefaultType directive for the current request. If not set "text/plain" is returned.

```
$content_type = $r->default_type();
```

obj: $r ( "Apache2::RequestRec object" )

The current request

ret: $content_type ( string )

The default type

since: 2.0.00

removed from the "httpd" API in version 2.3.2

"dir_config"

"$r->dir_config()" provides an interface for the per-directory variable specified by the "PerlSetVar" and "PerlAddVar" directives, and also can be manipulated via the "APR::Table" methods.

```
$table  = $r->dir_config();
$value  = $r->dir_config($key);
@values = $r->dir_config->get($key);
$r->dir_config($key, $val);
```

obj: $r ( "Apache2::RequestRec object" )

opt arg2: $key ( string )

Key string

opt arg3: $val ( string )

Value string

ret: ...

Depends on the passed arguments, see further discussion

since: 2.0.00

The keys are case-insensitive.

```
$apr_table = $r->dir_config();
```

dir_config() called in a scalar context without the $key argument returns a HASH reference blessed into the "APR::Table" class. This object can be manipulated via the "APR::Table" methods. For available methods see the "APR::Table" manpage.

```
$value = $r->dir_config($key);
```

If the $key argument is passed in the scalar context only a single value will be returned. Since the table preserves the insertion order, if there is more than one value for the same key, the oldest value associated with the desired key is returned. Calling in the scalar context is also much faster, as it'll stop searching the table as soon as the first match happens.

```
@values = $r->dir_config->get($key);
```

To receive a list of values you must use "get()" method from the "APR::Table" class.

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

If the $key and the $val arguments are used, the set() operation will happen: all existing values associated with the key $key (and the key itself) will be deleted and $value will be placed instead.

```
$r->dir_config($key => undef);
```

If $val is undef the unset() operation will happen: all existing values associated with the key $key (and the key itself) will be deleted.

"document_root"

Retrieve the document root for this server

```
$docroot = $r->document_root();
$docroot = $r->document_root($new_root);
```

obj: $r ( "Apache2::RequestRec object" )

The current request

opt arg1: $new_root

Sets the document root to a new value only for the duration of the current request.

Note the limited functionality under threaded MPMs.

ret: $docroot ( string )

The document root

since: 2.0.00

"get_handlers"

Returns a reference to a list of handlers enabled for a given phase.

```
$handlers_list = $r->get_handlers($hook_name);
```

obj: $r ( "Apache2::RequestRec object" )

arg1: $hook_name ( string )

a string representing the phase to handle (e.g. "PerlLogHandler")

ret: $handlers_list (ref to an ARRAY of CODE refs)

a list of handler subroutines CODE references

since: 2.0.00

See also: "$s->add_config"

For example:

A list of handlers configured to run at the response phase:

```
my @handlers = @{ $r->get_handlers('PerlResponseHandler') || [] };
```

"get_limit_req_body"

Return the limit on bytes in request msg body

```
$limit = $r->get_limit_req_body();
```

obj: $r ( "Apache2::RequestRec object" )

The current request

ret: $limit (integer)

the maximum number of bytes in the request msg body

since: 2.0.00

"get_server_name"

Get the current request's server name

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

obj: $r ( "Apache2::RequestRec object" )

The current request

ret: $server ( string )

the server name

since: 2.0.00

For example, consruct a hostport string:

```
use Apache2::RequestUtil ();
my $hostport = join ':', $r->get_server_name, $r->get_server_port;
```

"get_server_port"

Get the current server port

$port = $r->get_server_port();

obj: $r ( "Apache2::RequestRec object" )

The current request

ret: $port ( integer )

The server's port number

since: 2.0.00

For example, consruct a hostport string:

use Apache2::RequestUtil ();

my $hostport = join ':', $r->get_server_name, $r->get_server_port;

"get_status_line"

Return the "Status-Line" for a given status code (excluding the HTTP-Version field).

$status_line = Apache2::RequestUtil::get_status_line($status);

arg1: $status (integer)

The HTTP status code

ret: $status_line ( string )

The Status-Line

If an invalid or unknown status code is passed, "500 Internal Server Error" will be

returned.

since: 2.0.00

For example:

use Apache2::RequestUtil ();

print Apache2::RequestUtil::get_status_line(400);

will print:

400 Bad Request

"is_initial_req"

Determine whether the current request is the main request or a sub-request

$is_initial = $r->is_initial_req();

obj: $r ( "Apache2::RequestRec object" )

A request or a sub-request object

ret: $is_initial ( boolean )

If true -- it's the main request, otherwise it's a sub-request

since: 2.0.00

"is_perl_option_enabled"

check whether a directory level "PerlOptions" flag is enabled or not.

```
$result = $r->is_perl_option_enabled($flag);
```

obj: $r ( "Apache2::RequestRec object" )

arg1: $flag ( string )

ret: $result ( boolean )

since: 2.0.00

For example to check whether the "SetupEnv" option is enabled for the current request (which can be disabled with "PerlOptions -SetupEnv") and populate the environment variables table if disabled:

```
$r->subprocess_env unless $r->is_perl_option_enabled('SetupEnv');
```

See also: PerlOptions and the equivalent function for server level PerlOptions flags.

"location"

Get the path of the <Location> section from which the current "Perl*Handler" is being called.

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

obj: $r ( "Apache2::RequestRec object" )

ret: $location ( string )

since: 2.0.00

"location_merge"

Merge a given "<Location>" container into the current request object:

```
$ret = $r->location_merge($location);
```

obj: $r ( "Apache2::RequestRec object" )

arg1: $location ( string )

The argument in a "<Location>" section. For example to merge a container:

```
<Location /foo>
    ...
</Location>
```

that argument will be /foo

ret: $ret ( boolean )

a true value if the merge was successful (i.e. the request $location match was found), otherwise false.

since: 2.0.00

Useful for insertion of a configuration section into a custom "Apache2::RequestRec" object, created via the "Apache2::RequestRec->new()" method. See for example the Command Server protocol example.

"new"

Create a new "Apache2::RequestRec" object.

```
$r = Apache2::RequestRec->new($c);
$r = Apache2::RequestRec->new($c, $pool);
```

obj: "Apache2::RequestRec" ( "Apache2::RequestRec class name" )

arg1: $c ("Apache2::Connection object")

opt arg2: $pool

If no $pool argument is passed, "$c->pool" is used. That means that the created "Apache2::RequestRec" object will be valid as long as the connection object is valid.

ret: $r ( "Apache2::RequestRec object" )

since: 2.0.00

It's possible to reuse the HTTP framework features outside the familiar HTTP request cycle. It's possible to write your own full or partial HTTP implementation without needing a running Apache server. You will need the "Apache2::RequestRec" object in order to be able to reuse the rich functionality supplied via this object.

See for example the Command Server protocol example which reuses HTTP AAA model under non-HTTP protocol.

"no_cache"

Add/remove cache control headers:

```
$prev_no_cache = $r->no_cache($boolean);
```

obj: $r ( "Apache2::RequestRec object" )

arg1: $boolean ( boolean )

A true value sets the "no_cache" request record member to a true value and inserts:

```
Pragma: no-cache
Cache-control: no-cache
```

into the response headers, indicating that the data being returned is volatile and the client should not cache it.

A false value unsets the "no_cache" request record member and the mentioned headers if they were previously set.

ret: $prev_no_cache ( boolean )

Should you care, the "no_cache" request record member value prior to the change is returned.

since: 2.0.00

This method should be invoked before any response data has been sent out.

"pnotes"

Share Perl variables between Perl HTTP handlers

```
# to share variables by value and not reference, $val should be a lexical.
$old_val  = $r->pnotes($key => $val);
$val      = $r->pnotes($key);
$hash_ref = $r->pnotes();
```

Note: sharing variables really means it. The variable is not copied. Only its reference count is incremented. If it is changed after being put in pnotes that change also affects the stored value. The following example illustrates the effect:

```
my $v=1;                    my $v=1;
$r->pnotes( 'v'=>$v );      $r->pnotes->{v}=$v;
$v++;                       $v++;
my $x=$r->pnotes('v');      my $x=$r->pnotes->{v};
```

In both cases $x is 2 not 1. See also "Apache2::SafePnotes" on CPAN.

There has been a lot of discussion advocating for pnotes sharing variables by value and not reference. Sharing by reference can create 'spooky action at a distance' effects when the sharing is assumed to share a copy of the value. Tim Bunce offers the following summary and suggestion for sharing by value.

What's wrong with this code:

```
sub foo {
    my ($r, $status, $why) = @_;
    $r->pnotes('foo', ($why) ? "$status:$why" : $status);
    return;
}
```

Nothing, except it doesn't work as expected due to this pnotes bug: If the same code is called in a sub-request then the pnote of $r->prev is magically updated at a distance to the same value!

Try to explain why that is to anyone not deeply familiar with perl internals!

The fix is to avoid pnotes taking a ref to the invisible op_targ embededed in the code by

passing a simple lexical variable as the actual argument. That can be done in-line like this:

```
 sub mark_as_internally_redirected {
     my ($r, $status, $why) = @_;
     $r->pnotes('foo', my $tmp = (($why) ? "$status:$why" : $status));
     return;
 }
```

obj: $r ( "Apache2::RequestRec object" )

opt arg1: $key ( string )

A key value

opt arg2: $val ( SCALAR )

Any scalar value (e.g. a reference to an array)

ret: (3 different possible values)

if both, $key and $val are passed the previous value for $key is returned if such existed, otherwise "undef" is returned.

if only $key is passed, the current value for the given key is returned.

if no arguments are passed, a hash reference is returned, which can then be directly accessed without going through the "pnotes()" interface.

since: 2.0.00

This method provides functionality similar to ("Apache2::RequestRec::notes"), but values can be any Perl variables. That also means that it can be used only between Perl modules.

The values get reset automatically at the end of each HTTP request.

Examples:

Set a key/value pair:

```
 $r->pnotes(foo => [1..5]);
```

Get the value:

```
 $val = $r->pnotes("foo");
```

$val now contains an array ref containing 5 elements (1..5).

Now change the existing value:

```
 $old_val = $r->pnotes(foo => ['a'..'c']);
 $val = $r->pnotes("foo");
```

$old_val now contains an array ref with 5 elements (1..5) and $val contains an array ref with 3 elements 'a', 'b', 'c'.

Alternatively you can access the hash reference with all pnotes values:

```
$pnotes = $r->pnotes;
```

Now we can read what's in there for the key foo:

```
$val = $pnotes->{foo};
```

and as before $val still gives us an array ref with 3 elements 'a', 'b', 'c'.

Now we can add elements to it:

```
push @{ $pnotes{foo} }, 'd'..'f';
```

and we can try to retrieve them using the hash and non-hash API:

```
$val1 = $pnotes{foo};
$val2 = $r->pnotes("foo");
```

Both $val1 and $val2 contain an array ref with 6 elements (letters 'a' to 'f').

Finally to reset an entry you could just assign "undef" as a value:

```
$r->pnotes(foo => undef);
```

but the entry for the key foo still remains with the value "undef". If you really want to completely remove it, use the hash interface:

```
delete $r->pnotes->{foo};
```

"psignature"

Get HTML describing the address and (optionally) admin of the server.

```
$sig = $r->psignature($prefix);
```

obj: $r ( "Apache2::RequestRec" )

arg1: $prefix ( string )

Text which is prepended to the return value

ret: $sig ( string )

HTML text describing the server. Note that depending on the value of the "ServerSignature" directive, the function may return the address, including the admin information or nothing at all.

since: 2.0.00

"request"

Get/set the ( "Apache2::RequestRec object" ) object for the current request.

```
$r = Apache2::RequestUtil->request;
Apache2::RequestUtil->request($new_r);
```

obj: "Apache2" (class name)

The Apache class name

opt arg1: $new_r ( "Apache2::RequestRec object" )

ret: $r ( "Apache2::RequestRec object" )

since: 2.0.00

The get-able part of this method is only available if "PerlOptions +GlobalRequest" is in effect or if "Apache2->request($new_r)" was called earlier. So instead of setting "PerlOptions +GlobalRequest", one can set the global request from within the handler.

"push_handlers"

Add one or more handlers to a list of handlers to be called for a given phase.

```
$ok = $r->push_handlers($hook_name => \&handler);
$ok = $r->push_handlers($hook_name => ['Foo::Bar::handler', \&handler2]);
```

obj: $r ( "Apache2::RequestRec object" )

arg1: $hook_name ( string )

the phase to add the handlers to

arg2: $handlers ( CODE ref or SUB name or an ARRAY ref )

a single handler CODE reference or just a name of the subroutine (fully qualified unless defined in the current package).

if more than one passed, use a reference to an array of CODE refs and/or subroutine names.

ret: $ok ( boolean )

returns a true value on success, otherwise a false value

since: 2.0.00

See also: "$s->add_config"

Note that to push input/output filters you have to use "Apache2::Filter" methods: "add_input_filter" and "add_output_filter".

Examples:

A single handler:

```
$r->push_handlers(PerlResponseHandler => \&handler);
```

Multiple handlers:

```
$r->push_handlers(PerlFixupHandler => ['Foo::Bar::handler', \&handler2]);
```

Anonymous functions:

```
$r->push_handlers(PerlLogHandler => sub { return Apache2::Const::OK });
```

"set_basic_credentials"

Populate the incoming request headers table ("headers_in") with authentication headers for

Basic Authorization as if the client has submitted those in first place:

```
$r->set_basic_credentials($username, $password);
```

obj: $r ( "Apache2::RequestRec object" )

arg1: $username ( string )

arg2: $password ( string )

ret: no return value

since: 2.0.00

See for example the Command Server protocol example which reuses HTTP AAA model under non-HTTP protocol.

"set_handlers"

Set a list of handlers to be called for a given phase. Any previously set handlers are forgotten.

```
$ok = $r->set_handlers($hook_name => \&handler);
$ok = $r->set_handlers($hook_name => ['Foo::Bar::handler', \&handler2]);
$ok = $r->set_handlers($hook_name => []);
$ok = $r->set_handlers($hook_name => undef);
```

obj: $r ( "Apache2::RequestRec object" )

arg1: $hook_name ( string )

the phase to set the handlers in

arg2: $handlers (CODE ref or SUB name or an ARRAY ref)

a reference to a single handler CODE reference or just a name of the subroutine (fully qualified unless defined in the current package).

if more than one passed, use a reference to an array of CODE refs and/or subroutine names.

if the argument is "undef" or "[]" the list of handlers is reset to zero.

ret: $ok ( boolean )

returns a true value on success, otherwise a false value

since: 2.0.00

See also: "$s->add_config"

Examples:

A single handler:

```
$r->set_handlers(PerlResponseHandler => \&handler);
```

Multiple handlers:

```
  $r->set_handlers(PerlFixupHandler => ['Foo::Bar::handler', \&handler2]);
```

Anonymous functions:

```
  $r->set_handlers(PerlLogHandler => sub { return Apache2::Const::OK });
```

Reset any previously set handlers:

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

or

```
  $r->set_handlers(PerlCleanupHandler => undef);
```

"slurp_filename"

Slurp the contents of "$r->filename":

```
  $content_ref = $r->slurp_filename($tainted);
```

obj: $r ( "Apache2::RequestRec object" )

arg1: $tainted (number)

If the server is run under the tainting mode ("-T") which we hope you do, by default the returned data is tainted. If an optional $tainted flag is set to zero, the data will be marked as non-tainted.

Do not set this flag to zero unless you know what you are doing, you may create a security hole in your program if you do. For more information see the perlsec manpage.

If you wonder why this option is available, it is used internally by the "ModPerl::Registry" handler and friends, because the CGI scripts that it reads are considered safe (you could just as well "require()" them).

ret: $content_ref ( SCALAR ref )

A reference to a string with the contents

excpt: "APR::Error"

Possible error codes could be: "APR::Const::EACCES" (permission problems), "APR::Const::ENOENT" (file not found), and others. For checking such error codes, see the documentation for, for example, "APR::Status::is_EACCES" and "APR::Status::is_ENOENT".

since: 2.0.00

Note that if you assign to "$r->filename" you need to update its stat record.

## See Also

mod_perl 2.0 documentation.

## Copyright

Version 2.0.

## Authors

The mod_perl development team and numerous contributors.