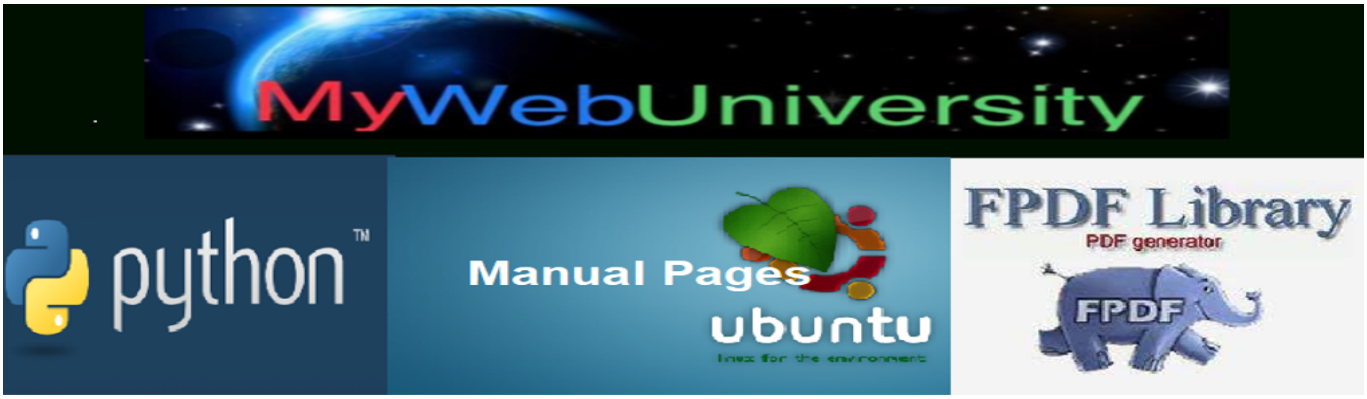

Rocky Enterprise Linux 9.2 Manual Pages on command 'Apache::TestRequest.3pm'

$ man Apache::TestRequest.3pm

Apache::TestRequest(3pm) User Contributed Perl Documentation Apache::TestRequest(3pm)

NAME

Apache::TestRequest - Send requests to your Apache test server

SYNOPSIS

```
use Apache::Test qw(ok have_lwp);
use Apache::TestRequest qw(GET POST);
use Apache::Constants qw(HTTP_OK);
plan tests => 1, have_lwp;
my $res = GET '/test.html';
ok $res->code == HTTP_OK, "Request is ok";
```

DESCRIPTION

Apache::TestRequest provides convenience functions to allow you to make requests to your Apache test server in your test scripts. It subclasses "LWP::UserAgent", so that you have access to all if its methods, but also exports a number of useful functions likely useful for majority of your test requests. Users of the old "Apache::test" (or "Apache::testold") module, take note! Herein lie most of the functions you'll need to use to replace "Apache::test" in your test suites.

Each of the functions exported by "Apache::TestRequest" uses an "LWP::UserAgent" object to submit the request and retrieve its results. The return value for many of these functions is an HTTP::Response object. See HTTP::Response for documentation of its methods, which you can use in your tests. For example, use the "code()" and "content()" methods to test the response code and content of your request. Using "GET", you can perform a couple of tests using these methods like this:

```
use Apache::Test qw(ok have_lwp);
use Apache::TestRequest qw(GET POST);
use Apache::Constants qw(HTTP_OK);
plan tests => 2, have_lwp;
my $uri = "/test.html?foo=1&bar=2";
my $res = GET $uri;
ok $res->code == HTTP_OK, "Check that the request was OK";
ok $res->content eq "foo => 1, bar => 2", "Check its content";
```

Note that you can also use "Apache::TestRequest" with "Test::Builder" and its derivatives, including "Test::More":

```
use Test::More;
# ...
is $res->code, HTTP_OK, "Check that the request was OK";
is $res->content, "foo => 1, bar => 2", "Check its content";
```

CONFIGURATION FUNCTION

You can tell "Apache::TestRequest" what kind of "LWP::UserAgent" object to use for its convenience functions with "user_agent()". This function uses its arguments to construct an internal global "LWP::UserAgent" object that will be used for all subsequent requests made by the convenience functions. The arguments it takes are the same as for the "LWP::UserAgent" constructor. See the "LWP::UserAgent" documentation for a complete list.

The "user_agent()" function only creates the internal "LWP::UserAgent" object the first time it is called. Since this function is called internally by "Apache::TestRequest", you should always use the "reset" parameter to force it to create a new global "LWP::UserAgent" Object:

```
Apache::TestRequest::user_agent(reset => 1, %params);
```

"user_agent()" differs from "LWP::UserAgent->new" in two additional ways. First, it supports an additional parameter, "keep_alive", which enables connection persistence, where the same connection is used to process multiple requests (and, according to the "LWP::UserAgent" documentation, has the effect of loading and enabling the new experimental HTTP/1.1 protocol module).

And finally, the semantics of the "requests_redirectable" parameter is different than for "LWP::UserAgent" in that you can pass it a boolean value as well as an array for "LWP::UserAgent". To force "Apache::TestRequest" not to follow redirects in any of its

convenience functions, pass a false value to "requests_redirectable":

```
Apache::TestRequest::user_agent(reset => 1,
                                requests_redirectable => 0);
```

If LWP is not installed, then you can still pass in an array reference as "LWP::UserAgent" expects. "Apache::TestRequest" will examine the array and allow redirects if the array contains more than one value or if there is only one value and that value is not "POST":

```
# Always allow redirection.
my $redir = have_lwp() ? [qw(GET HEAD POST)] : 1;
Apache::TestRequest::user_agent(reset => 1,
                                requests_redirectable => $redir);
```

But note that redirection will not work with "POST" unless LWP is installed. It's best, therefore, to check "have_lwp" before running tests that rely on a redirection from "POST".

Sometimes it is desirable to have "Apache::TestRequest" remember cookies sent by the pages you are testing and send them back to the server on subsequent requests. This is especially necessary when testing pages whose functionality relies on sessions or the presence of preferences stored in cookies.

By default, "LWP::UserAgent" does not remember cookies between requests. You can tell it to remember cookies between request by adding:

```
Apache::TestRequest::user_agent(cookie_jar => {});
```

before issuing the requests.

FUNCTIONS

"Apache::TestRequest" exports a number of functions that will likely prove convenient for use in the majority of your request tests.

Optional Parameters

Each function also takes a number of optional arguments.

redirect_ok

By default a request will follow redirects retrieved from the server. To prevent this behavior, pass a false value to a "redirect_ok" parameter:

```
my $res = GET $uri, redirect_ok => 0;
```

Alternately, if all of your tests need to disable redirects, tell "Apache::TestRequest" to use an "LWP::UserAgent" object that disables redirects:

```
Apache::TestRequest::user_agent( reset => 1,
```

requests_redirectable => 0);

cert

If you need to force an SSL request to use a particular SSL certificate, pass the name of the certificate via the "cert" parameter:

```
my $res = GET $uri, cert => 'my_cert';
```

content

If you need to add content to your request, use the "content" parameter:

```
my $res = GET $uri, content => 'hello world!';
```

filename

The name of a local file on the file system to be sent to the Apache test server via "UPLOAD()" and its friends.

The Functions

GET

```
my $res = GET $uri;
```

Sends a simple GET request to the Apache test server. Returns an "HTTP::Response" object.

You can also supply additional headers to be sent with the request by adding their name/value pairs after the "url" parameter, for example:

```
my $res = GET $url, 'Accept-Language' => 'de,en-us,en;q=0.5';
```

GET_STR

A shortcut function for "GET($uri)->as_string".

GET_BODY

A shortcut function for "GET($uri)->content".

GET_BODY_ASSERT

Use this function when your test is outputting content that you need to check, and you want to make sure that the request was successful before comparing the contents of the request. If the request was unsuccessful, "GET_BODY_ASSERT" will return an error message. Otherwise it will simply return the content of the request just as "GET_BODY" would.

GET_OK

A shortcut function for "GET($uri)->is_success".

GET_RC

A shortcut function for "GET($uri)->code".

GET_HEAD

Throws out the content of the request, and returns the string representation of the

request. Since the body has been thrown out, the representation will consist solely of the headers. Furthermore, "GET_HEAD" inserts a "#" at the beginning of each line of the return string, so that the contents are suitable for printing to STDERR during your tests without interfering with the workings of "Test::Harness".

HEAD

```
my $res = HEAD $uri;
```

Sends a HEAD request to the Apache test server. Returns an "HTTP::Response" object.

HEAD_STR

A shortcut function for "HEAD($uri)->as_string".

HEAD_BODY

A shortcut function for "HEAD($uri)->content". Of course, this means that it will likely return nothing.

HEAD_BODY_ASSERT

Use this function when your test is outputting content that you need to check, and you want to make sure that the request was successful before comparing the contents of the request. If the request was unsuccessful, "HEAD_BODY_ASSERT" will return an error message. Otherwise it will simply return the content of the request just as "HEAD_BODY" would.

HEAD_OK

A shortcut function for "GET($uri)->is_success".

HEAD_RC

A shortcut function for "GET($uri)->code".

HEAD_HEAD

Throws out the content of the request, and returns the string representation of the request. Since the body has been thrown out, the representation will consist solely of the headers. Furthermore, "GET_HEAD" inserts a "#" at the beginning of each line of the return string, so that the contents are suitable for printing to STDERR during your tests without interfering with the workings of "Test::Harness".

PUT

```
my $res = PUT $uri;
```

Sends a simple PUT request to the Apache test server. Returns an "HTTP::Response" object.

PUT_STR

A shortcut function for "PUT($uri)->as_string".

PUT_BODY

A shortcut function for "PUT($uri)->content".

PUT_BODY_ASSERT

Use this function when your test is outputting content that you need to check, and you want to make sure that the request was successful before comparing the contents of the request. If the request was unsuccessful, "PUT_BODY_ASSERT" will return an error message. Otherwise it will simply return the content of the request just as "PUT_BODY" would.

PUT_OK

A shortcut function for "PUT($uri)->is_success".

PUT_RC

A shortcut function for "PUT($uri)->code".

PUT_HEAD

Throws out the content of the request, and returns the string representation of the request. Since the body has been thrown out, the representation will consist solely of the headers. Furthermore, "PUT_HEAD" inserts a "#" at the beginning of each line of the return string, so that the contents are suitable for printing to STDERR during your tests without interfering with the workings of "Test::Harness".

POST

```
  my $res = POST $uri, [ arg => $val, arg2 => $val ];
```

Sends a POST request to the Apache test server and returns an "HTTP::Response" object. An array reference of parameters passed as the second argument will be submitted to the Apache test server as the POST content. Parameters corresponding to those documented in Optional Parameters can follow the optional array reference of parameters, or after $uri.

To upload a chunk of data, simply use:

```
  my $res = POST $uri, content => $data;
```

POST_STR

A shortcut function for "POST($uri, @args)->content".

POST_BODY

A shortcut function for "POST($uri, @args)->content".

POST_BODY_ASSERT

Use this function when your test is outputting content that you need to check, and you want to make sure that the request was successful before comparing the contents of the request. If the request was unsuccessful, "POST_BODY_ASSERT" will return an error message. Otherwise it will simply return the content of the request just as "POST_BODY" would.

POST_OK

A shortcut function for "POST($uri, @args)->is_success".

POST_RC

A shortcut function for "POST($uri, @args)->code".

POST_HEAD

Throws out the content of the request, and returns the string representation of the request. Since the body has been thrown out, the representation will consist solely of the headers. Furthermore, "POST_HEAD" inserts a "#" at the beginning of each line of the return string, so that the contents are suitable for printing to STDERR during your tests without interfering with the workings of "Test::Harness".

UPLOAD

```
my $res = UPLOAD $uri, \@args, filename => $filename;
```

Sends a request to the Apache test server that includes an uploaded file. Other POST parameters can be passed as a second argument as an array reference. "Apache::TestRequest" will read in the contents of the file named via the "filename" parameter for submission to the server. If you'd rather, you can submit use the "content" parameter instead of "filename", and its value will be submitted to the Apache server as file contents:

```
my $res = UPLOAD $uri, undef, content => "This is file content";
```

The name of the file sent to the server will simply be "b". Note that in this case, you cannot pass other POST arguments to "UPLOAD()" -- they would be ignored.

UPLOAD_BODY

A shortcut function for "UPLOAD($uri, @params)->content".

UPLOAD_BODY_ASSERT

Use this function when your test is outputting content that you need to check, and you want to make sure that the request was successful before comparing the contents of the request. If the request was unsuccessful, "UPLOAD_BODY_ASSERT" will return an error message. Otherwise it will simply return the content of the request just as "UPLOAD_BODY" would.

OPTIONS

```
my $res = OPTIONS $uri;
```

Sends an "OPTIONS" request to the Apache test server. Returns an "HTTP::Response" object with the Allow header, indicating which methods the server supports. Possible methods

include "OPTIONS", "GET", "HEAD" and "POST". This function thus can be useful for testing what options the Apache server supports. Consult the HTTPD 1.1 specification, section 9.2, at http://www.faqs.org/rfcs/rfc2616.html for more information.

URL Manipulation Functions

"Apache::TestRequest" also includes a few helper functions to aid in the creation of urls used in the functions above.

"module2path"

```
$path = Apache::TestRequest::module2path($module_name);
```

Convert a module name to a path, safe for use in the various request methods above. e.g. "::" can't be used in URLs on win32. For example:

```
$path = Apache::TestRequest::module2path('Foo::Bar');
```

returns:

```
/Foo__Bar
```

"module2url"

```
$url = Apache::TestRequest::module2url($module);
$url = Apache::TestRequest::module2url($module, \%options);
```

Convert a module name to a full URL including the current configurations "hostname:port" and sets "module" accordingly.

```
$url = Apache::TestRequest::module2url('Foo::Bar');
```

returns:

```
http://$hostname:$port/Foo__Bar
```

The default scheme used is "http". You can override this by passing your preferred scheme into an optional second param. For example:

```
$module = 'MyTestModule::TestHandler';
$url = Apache::TestRequest::module2url($module, {scheme => 'https'});
```

returns:

```
https://$hostname:$port/MyTestModule__TestHandler
```

You may also override the default path with a path of your own:

```
$module = 'MyTestModule::TestHandler';
$url = Apache::TestRequest::module2url($module, {path => '/foo'});
```

returns:

```
http://$hostname:$port/foo
```

The following environment variables can affect the behavior of "Apache::TestRequest":

APACHE_TEST_PRETEND_NO_LWP

If the environment variable "APACHE_TEST_PRETEND_NO_LWP" is set to a true value, "Apache::TestRequest" will pretend that LWP is not available so one can test whether the test suite will survive on a system which doesn't have libwww-perl installed.

APACHE_TEST_HTTP_09_OK

If the environment variable "APACHE_TEST_HTTP_09_OK" is set to a true value, "Apache::TestRequest" will allow HTTP/0.9 responses from the server to proceed. The default behavior is to die if the response protocol is not either HTTP/1.0 or HTTP/1.1.

SEE ALSO

Apache::Test is the main Apache testing module. Use it to set up your tests, create a plan, and to ensure that you have the Apache version and modules you need.

Use Apache::TestMM in your Makefile.PL to set up your distribution for testing.

AUTHOR

Doug MacEachern with contributions from Geoffrey Young, Philippe M. Chiasson, Stas Bekman and others. Documentation by David Wheeler.

Questions can be asked at the test-dev <at> httpd.apache.org list. For more information see: http://httpd.apache.org/test/ and http://perl.apache.org/docs/general/testing/testing.html.