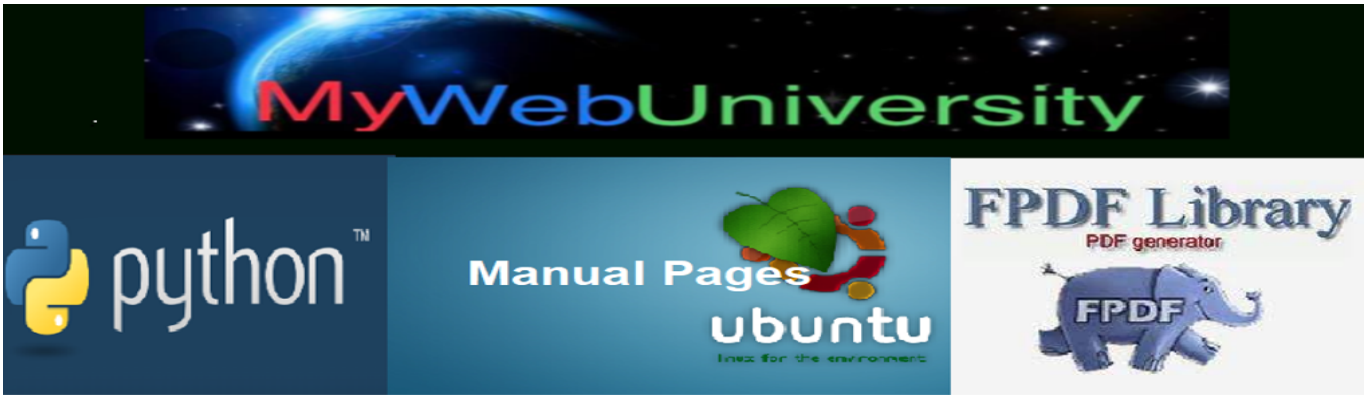

Rocky Enterprise Linux 9.2 Manual Pages on command 'Apache2::Access.3pm'

$ man Apache2::Access.3pm

libapache2-mod-perl2-2.0.12::doUseraClibapache2-mod-perl2-2.0.12::docs::api::Apache2::Access(3pm)

NAME

Apache2::Access - A Perl API for Apache request object: Access, Authentication and Authorization.

Synopsis

```
use Apache2::Access ();

# allow only GET method
$r->allow_methods(1, qw(GET));

# Apache Options value
$options = $r->allow_options();

# Apache AllowOverride value
$allow_override = $r->allow_overrides();

# which Options are allowed by AllowOverride (since Apache 2.2)
$allow_override_opts = $r->allow_override_opts();

# auth name ("foo bar")
$auth_name = $r->auth_name();
```

```
# auth type
$auth_type = $r->auth_type();
$r->auth_type("Digest");

# Basic authentication process
my ($rc, $passwd) = $r->get_basic_auth_pw();

# the login name of the remote user (RFC1413)
$remote_logname = $r->get_remote_logname();

# dynamically figure out which auth has failed
$r->note_auth_failure();

# note Basic auth failure
$r->note_basic_auth_failure();

# note Digest auth failure
$r->note_digest_auth_failure();

# Apache Request value(s)
$requires = $r->requires();

# Apache Satisfy value (as a number)
$satisfy = $r->satisfies();

# check whether some auth is configured
$need_auth = $r->some_auth_required();
```

Description

The API provided by this module deals with access, authentication and authorization phases.

"Apache2::Access" extends "Apache2::RequestRec".

API

"Apache2::Access" provides the following functions and/or methods:

"allow_methods"

Specify which HTTP methods are allowed

```
$r->allow_methods($reset);
$r->allow_methods($reset, @methods);
```

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

The current request

arg1: $reset (boolean)

If a true value is passed all the previously allowed methods are removed. Otherwise the list is left intact.

opt arg2: @methods (array of strings)

a list of HTTP methods to be allowed (e.g. "GET" and "POST")

ret: no return value

since: 2.0.00

For example: here is how to allow only "GET" and "POST" methods, regardless to what was the previous setting:

```
$r->allow_methods(1, qw(GET POST));
```

"allow_options"

Retrieve the value of "Options" for this request

```
$options = $r->allow_options();
```

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

The current request

ret: $options (integer)

the "Options" bitmask. Normally used with bitlogic operators against "Apache2::Const :options constants".

since: 2.0.00

For example if the configuration for the current request was:

```
Options None
Options Indexes FollowSymLinks
```

The following applies:

```
use Apache2::Const -compile => qw(:options);
$r->allow_options & Apache2::Const::OPT_INDEXES;   # TRUE
$r->allow_options & Apache2::Const::OPT_SYM_LINKS; # TRUE
$r->allow_options & Apache2::Const::OPT_EXECCGI;   # FALSE
```

"allow_overrides"

Retrieve the value of "AllowOverride" for this request

```
$allow_override = $r->allow_overrides();
```

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

The current request

ret: $allow_override (integer)

the "AllowOverride" bitmask. Normally used with bitlogic operators against "Apache2::Const :override constants".

since: 2.0.00

For example if the configuration for the current request was:

```
AllowOverride AuthConfig
```

The following applies:

```
use Apache2::Const -compile => qw(:override);
$r->allow_overrides & Apache2::Const::OR_AUTHCFG; # TRUE
$r->allow_overrides & Apache2::Const::OR_LIMIT; # FALSE
```

"allow_override_opts"

Retrieve the bitmask of allowed "Options" set by "AllowOverride Options=..." for this request

```
$override_opts = $r->allow_override_opts();
```

Enabling single options was introduced in Apache 2.2. For Apache 2.0 this function returns "Apache2::Const::OPT_UNSET" | "Apache2::Const::OPT_ALL" | "Apache2::Const::OPT_INCNOEXEC" | "Apache2::Const::OPT_SYM_OWNER" | "Apache2::Const::OPT_MULTI", which corresponds to the default value (if not set) for Apache 2.2.

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

The current request

ret: $override_opts (integer)

the override options bitmask. Normally used with bitlogic operators against "Apache2::Const :options constants".

since: 2.0.3

For example if the configuration for the current request was:

```
AllowOverride Options=Indexes,ExecCGI
```

The following applies:

```
use Apache2::Const -compile => qw(:options);
$r->allow_override_opts & Apache2::Const::OPT_EXECCGI; # TRUE
$r->allow_override_opts & Apache2::Const::OPT_SYM_LINKS; # FALSE
```

"auth_name"

Get/set the current Authorization realm (the per directory configuration directive "AuthName"):

```
$auth_name = $r->auth_name();
$auth_name = $r->auth_name($new_auth_name);
```

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

The current request

opt arg1: $new_auth_name (string)

If $new_auth_name is passed a new "AuthName" value is set

ret: "$" (integer)

The current value of "AuthName"

since: 2.0.00

The "AuthName" directive creates protection realm within the server document space. To quote RFC 1945 "These realms allow the protected resources on a server to be partitioned into a set of protection spaces, each with its own authentication scheme and/or authorization database." The client uses the root URL of the server to determine which authentication credentials to send with each HTTP request. These credentials are tagged

with the name of the authentication realm that created them. Then during the authentication stage the server uses the current authentication realm, from "$r->auth_name", to determine which set of credentials to authenticate.

"auth_type"

Get/set the type of authorization required for this request (the per directory configuration directive "AuthType"):

```
$auth_type = $r->auth_type();
$auth_type = $r->auth_type($new_auth_type);
```

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

The current request

opt arg1: $new_auth_type (string)

If $new_auth_type is passed a new "AuthType" value is set

ret: "$" (integer)

The current value of "AuthType"

since: 2.0.00

Normally "AuthType" would be set to "Basic" to use the basic authentication scheme defined in RFC 1945, Hypertext Transfer Protocol -- HTTP/1.0. However, you could set to something else and implement your own authentication scheme.

"get_basic_auth_pw"

Get the password from the request headers

```
my ($rc, $passwd) = $r->get_basic_auth_pw();
```

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

The current request

ret1: $rc ("Apache2::Const constant")

"Apache2::Const::OK" if the $passwd value is set (and assured a correct value in "$r->user"); otherwise it returns an error code, either "Apache2::Const::HTTP_INTERNAL_SERVER_ERROR" if things are really confused, "Apache2::Const::HTTP_UNAUTHORIZED" if no authentication at all seemed to be in use, or "Apache2::Const::DECLINED" if there was authentication, but it wasn't "Basic" (in which case, the caller should presumably decline as well).

ret2: $ret (string)

The password as set in the headers (decoded)

since: 2.0.00

If "AuthType" is not set, this handler first sets it to "Basic".

"get_remote_logname"

Retrieve the login name of the remote user (RFC1413)

```
$remote_logname = $r->get_remote_logname();
```

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

The current request

ret: $remote_logname (string)

The username of the user logged in to the client machine, or an empty string if it could not be determined via RFC1413, which involves querying the client's identd or auth daemon.

since: 2.0.00

Do not confuse this method with "$r->user", which provides the username provided by the user during the server authentication.

"note_auth_failure"

Setup the output headers so that the client knows how to authenticate itself the next time, if an authentication request failed. This function works for both basic and digest authentication

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

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

The current request

ret: no return value

since: 2.0.00

This method requires "AuthType" to be set to "Basic" or "Digest". Depending on the setting it'll call either "$r->note_basic_auth_failure" or "$r->note_digest_auth_failure".

"note_basic_auth_failure"

Setup the output headers so that the client knows how to authenticate itself the next time, if an authentication request failed. This function works only for basic authentication

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

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

The current request

ret: no return value

since: 2.0.00

"note_digest_auth_failure"

Setup the output headers so that the client knows how to authenticate itself the next time, if an authentication request failed. This function works only for digest

authentication.

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

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

The current request

ret: no return value

since: 2.0.00

"requires"

Retrieve information about all of the requires directives for this request

```
$requires = $r->requires
```

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

The current request

ret: $requires (ARRAY ref)

Returns an array reference of hash references, containing information related to the "require" directive.

since: 2.0.00

This is normally used for access control.

For example if the configuration had the following require directives:

```
Require user  goo bar
Require group bar tar
```

this method will return the following datastructure:

```
[
  {
    'method_mask' => -1,
    'requirement' => 'user goo bar'
  },
  {
    'method_mask' => -1,
    'requirement' => 'group bar tar'
  }
];
```

The requirement field is what was passed to the "Require" directive. The method_mask field is a bitmask which can be modified by the "Limit" directive, but normally it can be safely ignored as it's mostly used internally. For example if the configuration was:

```
  Require user goo bar
  Require group bar tar
  <Limit POST>
    Require valid-user
  </Limit>
```

and the request method was "POST", "$r->requires" will return:

```
[
  {
    'method_mask' => -1,
    'requirement' => 'user goo bar'
  },
  {
    'method_mask' => -1,
    'requirement' => 'group bar tar'
  }
  {
```

```
    'method_mask' => 4,
    'requirement' => 'valid-user'
  }
];
```

But if the request method was "GET", it will return only:

```
[
  {
    'method_mask' => -1,
    'requirement' => 'user goo bar'
  },
  {
    'method_mask' => -1,
    'requirement' => 'group bar tar'
  }
];
```

As you can see Apache gives you the requirements relevant for the current request, so the method_mask is irrelevant.

It is also a good time to remind that in the general case, access control directives should not be placed within a <Limit> section. Refer to the Apache documentation for more information.

Using the same configuration and assuming that the request was of type POST, the following code inside an Auth handler:

```
my %require =
    map { my ($k, $v) = split /\s+/, $_->{requirement}, 2; ($k, $v||'') }
    @{ $r->requires };
```

will populate %require with the following pairs:

```
    'group' => 'bar tar',
    'user' => 'goo bar',
    'valid-user' => '',
```

"satisfies"

How the requires lines must be met. What's the applicable value of the "Satisfy" directive:

```
  $satisfy = $r->satisfies();
```

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

The current request

ret: $satisfy (integer)

How the requirements must be met. One of the "Apache2::Const :satisfy constants":

"Apache2::Const::SATISFY_ANY", "Apache2::Const::SATISFY_ALL" and "Apache2::Const::SATISFY_NOSPEC".

since: 2.0.00

See the documentation for the "Satisfy" directive in the Apache documentation.

"some_auth_required"

Can be used within any handler to determine if any authentication is required for the current request:

```
  $need_auth = $r->some_auth_required();
```

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

The current request

ret: $need_auth (boolean)

TRUE if authentication is required, FALSE otherwise

since: 2.0.00

See Also

mod_perl 2.0 documentation.

Copyright

Authors

The mod_perl development team and numerous contributors.