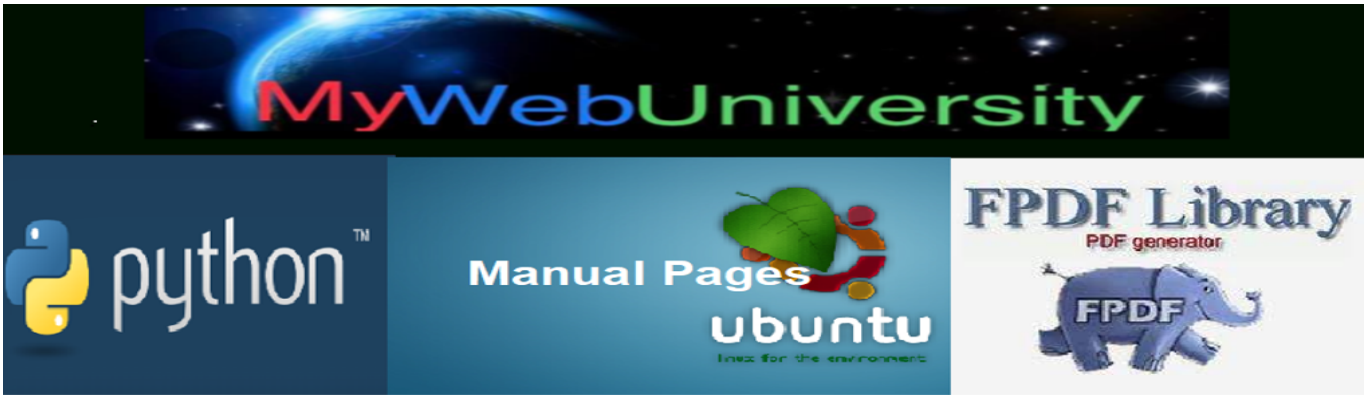


Rocky Enterprise Linux 9.2 Manual Pages on command 'Apache2::AuthCookie.3pm'

$ man Apache2::AuthCookie.3pm

Apache2::AuthCookie(3pm) User Contributed Perl Documentation Apache2::AuthCookie(3pm)

NAME

Apache2::AuthCookie - Perl Authentication and Authorization via cookies

VERSION

version 3.31

SYNOPSIS

Make sure your mod_perl is at least 2.0.0-RC5, with StackedHandlers, MethodHandlers,

Authen, and Authz compiled in.

```
# In httpd.conf or .htaccess:
PerlModule Sample::Apache2::AuthCookieHandler
PerlSetVar WhatEverPath /
PerlSetVar WhatEverLoginScript /login.pl
# use to alter how "require" directives are matched. Can be "Any" or "All".
# If its "Any", then you must only match Any of the "require" directives. If
# its "All", then you must match All of the require directives.
#
# Default: All
PerlSetVar WhatEverSatisfy Any
# The following line is optional - it allows you to set the domain
# scope of your cookie.  Default is the current domain.
PerlSetVar WhatEverDomain .yourdomain.com
# Use this to only send over a secure connection
PerlSetVar WhatEverSecure 1
```

```
# Use this if you want user session cookies to expire if the user
# doesn't request a auth-required or recognize_user page for some
# time period.  If set, a new cookie (with updated expire time)
# is set on every request.
PerlSetVar WhatEverSessionTimeout +30m
# to enable the HttpOnly cookie property, use HttpOnly.
# This is an MS extension.  See:
# http://msdn.microsoft.com/workshop/author/dhtml/httponly_cookies.asp
PerlSetVar WhatEverHttpOnly 1
# to enable the SameSite cookie property, set SameSite to "lax" or "strict".
# See: https://www.owasp.org/index.php/SameSite
PerlSetVar WhatEverSameSite strict
# Usually documents are uncached - turn off here
PerlSetVar WhatEverCache 1
# Use this to make your cookies persistent (+2 hours here)
PerlSetVar WhatEverExpires +2h
# Use to make AuthCookie send a P3P header with the cookie
# see http://www.w3.org/P3P/ for details about what the value
# of this should be
PerlSetVar WhatEverP3P "CP=\"...\""
# optional: enable decoding of intercepted GET/POST params:
PerlSetVar WhatEverEncoding UTF-8
# optional: enable decoding of httpd.conf "Requires" directives
PerlSetVar WhatEverRequiresEncoding UTF-8
# optional: enforce that the destination argument from the login form is
# local to the server
PerlSetVar WhatEverEnforceLocalDestination 1
# optional: specify a default destination for when the destination argument
# of the login form is invalid or unspecified
PerlSetVar WhatEverDefaultDestination /protected/user/
# These documents require user to be logged in.
<Location /protected>
 AuthType Sample::Apache2::AuthCookieHandler
```

```
 AuthName WhatEver
 PerlAuthenHandler Sample::Apache2::AuthCookieHandler->authenticate
 PerlAuthzHandler Sample::Apache2::AuthCookieHandler->authorize
 require valid-user
</Location>
# These documents don't require logging in, but allow it.
<FilesMatch "\.ok$">
 AuthType Sample::Apache2::AuthCookieHandler
 AuthName WhatEver
 PerlFixupHandler Sample::Apache2::AuthCookieHandler->recognize_user
</FilesMatch>
# This is the action of the login.pl script above.
<Files LOGIN>
 AuthType Sample::Apache2::AuthCookieHandler
 AuthName WhatEver
 SetHandler perl-script
 PerlResponseHandler Sample::Apache2::AuthCookieHandler->login
</Files>
```

DESCRIPTION

This module is for mod_perl version 2. If you are running mod_perl version 1, you should be using Apache::AuthCookie instead.

Apache2::AuthCookie allows you to intercept a user's first unauthenticated access to a protected document. The user will be presented with a custom form where they can enter authentication credentials. The credentials are posted to the server where AuthCookie verifies them and returns a session key.

The session key is returned to the user's browser as a cookie. As a cookie, the browser will pass the session key on every subsequent accesses. AuthCookie will verify the session key and re-authenticate the user.

All you have to do is write a custom module that inherits from AuthCookie. Your module is a class which implements two methods:

"authen_cred()"

Verify the user-supplied credentials and return a session key. The session key can be any string - often you'll use some string containing username, timeout info, and any

other information you need to determine access to documents, and append a one-way hash of those values together with some secret key.

"authen_ses_key()"

Verify the session key (previously generated by "authen_cred()", possibly during a previous request) and return the user ID. This user ID will be fed to "$r->user()" to set Apache's idea of who's logged in.

By using AuthCookie versus Apache's built-in AuthBasic you can design your own authentication system. There are several benefits.

1. The client doesn't *have* to pass the user credentials on every subsequent access. If you're using passwords, this means that the password can be sent on the first request only, and subsequent requests don't need to send this (potentially sensitive) information. This is known as "ticket-based" authentication.
2. When you determine that the client should stop using the credentials/session key, the server can tell the client to delete the cookie. Letting users "log out" is a notoriously impossible-to-solve problem of AuthBasic.
3. AuthBasic dialog boxes are ugly. You can design your own HTML login forms when you use AuthCookie.
4. You can specify the domain of a cookie using PerlSetVar commands. For instance, if your AuthName is "WhatEver", you can put the command

   PerlSetVar WhatEverDomain .yourhost.com

   into your server setup file and your access cookies will span all hosts ending in ".yourhost.com".
5. You can optionally specify the name of your cookie using the "CookieName" directive. For instance, if your AuthName is "WhatEver", you can put the command

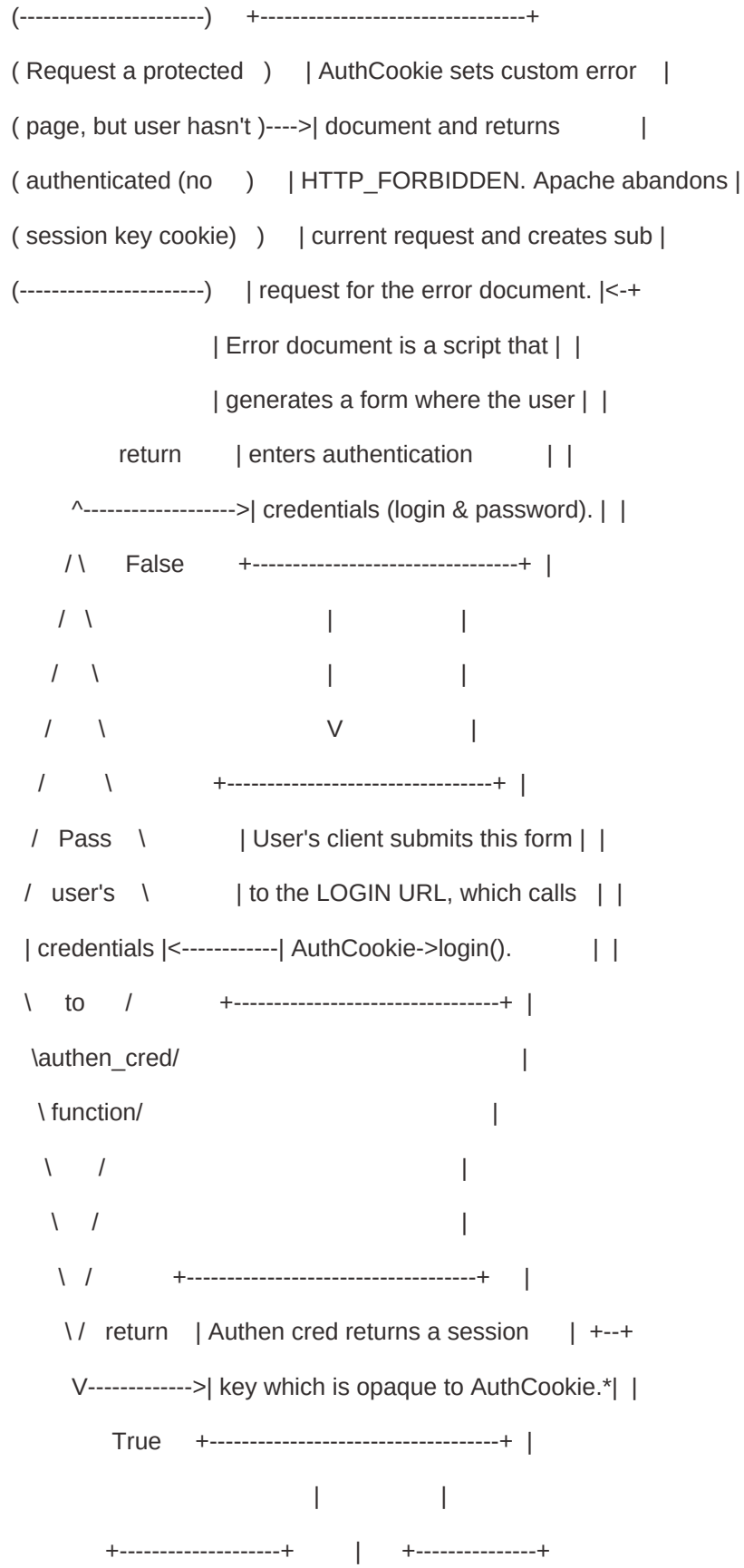
   PerlSetVar WhatEverCookieName MyCustomName

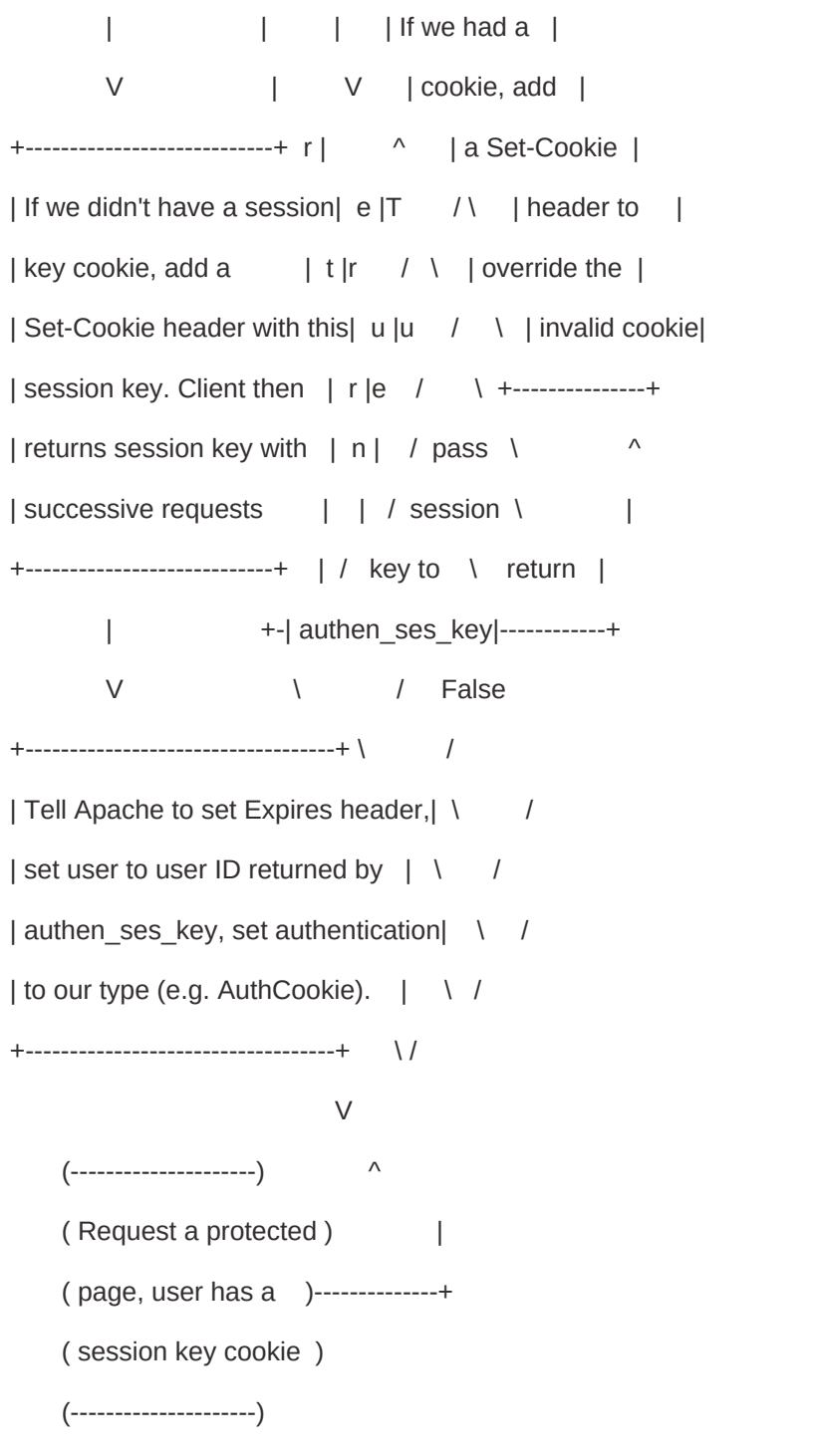
   into your server setup file and your cookies for this AuthCookie realm will be named MyCustomName. Default is AuthType_AuthName.
6. By default users must satisfy ALL of the "require" directives. If you want authentication to succeed if ANY "require" directives are met, use the "Satisfy" directive. For instance, if your AuthName is "WhatEver", you can put the command

   PerlSetVar WhatEverSatisfy Any

   into your server startup file and authentication for this realm will succeed if ANY of the "require" directives are met.

This is the flow of the authentication handler, less the details of the redirects. Two HTTP_MOVED_TEMPORARILY's are used to keep the client from displaying the user's credentials in the Location field. They don't really change AuthCookie's model, but they do add another round-trip request to the client.





* The session key that the client gets can be anything you want. For example, encrypted information about the user, a hash of the username and password (similar in function to Digest authentication), or the user name and password in plain text (similar in function to HTTP Basic authentication).

  The only requirement is that the authen_ses_key function that you create must be able to determine if this session_key is valid and map it back to the originally authenticated user ID.

authorize(): int

This will step through the "require" directives you've given for protected documents and make sure the user passes muster. The "require valid-user" and "require user joey-jojo" directives are handled for you. You can implement custom directives, such as "require species hamster", by defining a method called "species()" in your subclass, which will then be called. The method will be called as "$r->species($r, $args)", where $args is everything on your "require" line after the word "species". The method should return OK on success and HTTP_FORBIDDEN on failure.

get_satisfy(): string

Get the value of "${auth_name}Satisfy", or "all" if it is not set.

satisfy_is_valid(): bool

return true if the configured "${auth_name}Satisfy" is valid, false otherwise.

authen_cred(): string

You must define this method yourself in your subclass of "Apache2::AuthCookie". Its job is to create the session key that will be preserved in the user's cookie. The arguments passed to it are:

```
 sub authen_cred ($$\@) {
   my $self = shift;  # Package name (same as AuthName directive)
   my $r    = shift;  # Apache request object
   my @cred = @_;     # Credentials from login form
   ...blah blah blah, create a session key...
   return $session_key;
 }
```

The only limitation on the session key is that you should be able to look at it later and determine the user's username. You are responsible for implementing your own session key format. A typical format is to make a string that contains the username, an expiration time, whatever else you need, and an MD5 hash of all that data together with a secret key. The hash will ensure that the user doesn't tamper with the session key. More info in the Eagle book.

authen_ses_key($r, $session_key): string

You must define this method yourself in your subclass of Apache2::AuthCookie. Its job is to look at a session key and determine whether it is valid. If so, it returns the username of the authenticated user.

```
sub authen_ses_key ($$$) {
  my ($self, $r, $session_key) = @_;
  ...blah blah blah, check whether $session_key is valid...
  return $ok ? $username : undef;
}
```

Optionally, return an array of 2 or more items that will be passed to method custom_errors. It is the responsibility of this method to return the correct response to the main Apache module.

custom_errors($r,@_): int

Note: this interface is experimental.

This method handles the server response when you wish to access the Apache custom_response method. Any suitable response can be used. this is particularly useful when implementing 'by directory' access control using the user authentication information. i.e.

```
/restricted
    /one        user is allowed access here
    /two        not here
    /three      AND here
```

The authen_ses_key method would return a normal response when the user attempts to access 'one' or 'three' but return (NOT_FOUND, 'File not found') if an attempt was made to access subdirectory 'two'. Or, in the case of expired credentials, (AUTH_REQUIRED,'Your session has timed out, you must login again').

example 'custom_errors'

```
sub custom_errors {
  my ($self,$r,$CODE,$msg) = @_;
  # return custom message else use the server's standard message
  $r->custom_response($CODE, $msg) if $msg;
  return($CODE);
}
```

where CODE is a valid code from Apache2::Const

ENCODING AND CHARACTER SETS

Encoding

AuthCookie provides support for decoding POST/GET data if you tell it what the client encoding is. You do this by setting the "${auth_name}Encoding" setting in "httpd.conf".

E.g.:

```
PerlSetVar WhateEverEncoding UTF-8
# and you also need to arrange for charset=UTF-8 at the end of the
# Content-Type header with something like:
AddDefaultCharset UTF-8
```

Note that you can use charsets other than "UTF-8", however, you need to arrange for the browser to send the right encoding back to the server.

If you have turned on Encoding support by setting "${auth_name}Encoding", this has the following effects:

- The internal pure-perl params processing subclass will be used, even if libapreq2 is installed. libapreq2 does not have any support for encoding or unicode.
- POST/GET data intercepted by AuthCookie will be decoded to perl's internal format using "decode" in Encode.
- The value stored in "$r->user" will be encoded as bytes, not characters using the configured encoding name. This is because the value stored by mod_perl is a C API string, and not a perl string. You can use decoded_user() to get user string encoded using character semantics.

This does has some caveats:

- your authen_cred() and authen_ses_key() function is expected to return a decoded username, either by passing it through "decode()" in Encode, or, by turning on the UTF8 flag if appropriate.
- Due to the way HTTP works, cookies cannot contain non-ASCII characters. Because of this, if you are including the username in your generated session key, you will need to escape any non-ascii characters in the session key returned by authen_cred().
- Similarly, you must reverse this escaping process in authen_ses_key() and return a "decode()" in Encode decoded username. If your authen_cred() function already only generates ASCII-only session keys then you do not need to worry about any of this.
- The value stored in "$r->user" will be encoded using bytes semantics using the configured Encoding. If you want the decoded user value, use decoded_user() instead.

Requires

You can also specify what the charset is of the Apache "$r->requires" data is by setting "${auth_name}RequiresEncoding" in httpd.conf.

E.g.:

PerlSetVar WhatEverRequiresEncoding UTF-8

This will make it so that AuthCookie will decode your "requires" directives using the configured character set. You really only need to do this if you have used non-ascii characters in any of your "requires" directives in httpd.conf. e.g.:

requires user programm?r

EXAMPLE

For an example of how to use Apache2::AuthCookie, you may want to check out the test suite, which runs AuthCookie through a few of its paces. The documents are located in t/eg/, and you may want to peruse t/real.t to see the generated httpd.conf file (at the bottom of real.t) and check out what requests it's making of the server (at the top of real.t).

THE LOGIN SCRIPT

You will need to create a login script (called login.pl above) that generates an HTML form for the user to fill out. You might generate the page using a ModPerl::Registry script, a HTML::Mason component, an Apache handler, or perhaps even using a static HTML page. It's usually useful to generate it dynamically so that you can define the 'destination' field correctly (see below).

The following fields must be present in the form:

1. The ACTION of the form must be /LOGIN (or whatever you defined in your server configuration as handled by the ->login() method - see example in the SYNOPSIS section).
2. The various user input fields (username, passwords, etc.) must be named 'credential_0', 'credential_1', etc. on the form. These will get passed to your authen_cred() method.
3. You must define a form field called 'destination' that tells AuthCookie where to redirect the request after successfully logging in. Typically this value is obtained from "$r->prev->uri". See the login.pl script in t/eg/.

In addition, you might want your login page to be able to tell why the user is being asked to log in. In other words, if the user sent bad credentials, then it might be useful to display an error message saying that the given username or password are invalid. Also, it might be useful to determine the difference between a user that sent an invalid auth cookie, and a user that sent no auth cookie at all. To cope with these situations, AuthCookie will set "$r->subprocess_env('AuthCookieReason')" to one of the following

values.

no_cookie

The user presented no cookie at all. Typically this means the user is trying to log in for the first time.

bad_cookie

The cookie the user presented is invalid. Typically this means that the user is not allowed access to the given page.

bad_credentials

The user tried to log in, but the credentials that were passed are invalid.

You can examine this value in your login form by examining "$r->prev->subprocess_env('AuthCookieReason')" (because it's a sub-request).

Of course, if you want to give more specific information about why access failed when a cookie is present, your "authen_ses_key()" method can set arbitrary entries in "$r->subprocess_env".

THE LOGOUT SCRIPT

If you want to let users log themselves out (something that can't be done using Basic Auth), you need to create a logout script. For an example, see t/htdocs/docs/logout.pl.

Logout scripts may want to take advantage of AuthCookie's "logout()" method, which will set the proper cookie headers in order to clear the user's cookie. This usually looks like "$r->auth_type->logout($r);".

Note that if you don't necessarily trust your users, you can't count on cookie deletion for logging out. You'll have to expire some server-side login information too.

AuthCookie doesn't do this for you, you have to handle it yourself.

ABOUT SESSION KEYS

Unlike the sample AuthCookieHandler, you have you verify the user's login and password in "authen_cred()", then you do something like:

```
my $date = localtime;
my $ses_key = MD5->hexhash(join(';', $date, $PID, $PAC));
```

save $ses_key along with the user's login, and return $ses_key.

Now "authen_ses_key()" looks up the $ses_key passed to it and returns the saved login. I use Oracle to store the session key and retrieve it later, see the ToDo section below for some other ideas.

TO DO

? It might be nice if the logout method could accept some parameters that could make it easy to redirect the user to another URI, or whatever. I'd have to think about the options needed before I implement anything, though.

HISTORY

Originally written by Eric Bartley <bartley@purdue.edu>

versions 2.x were written by Ken Williams <ken@forum.swarthmore.edu>

COPYRIGHT

SEE ALSO

Apache2::AuthCookie::Base

SOURCE

The development version is on github at <https://https://github.com/mschout/apache-authcookie> and may be cloned from <git://https://github.com/mschout/apache-authcookie.git>

BUGS

Please report any bugs or feature requests on the bugtracker website <https://github.com/mschout/apache-authcookie/issues>

When submitting a bug or request, please include a test-file or a patch to an existing test-file that illustrates the bug or desired feature.

AUTHOR

Michael Schout <mschout@cpan.org>

COPYRIGHT AND LICENSE