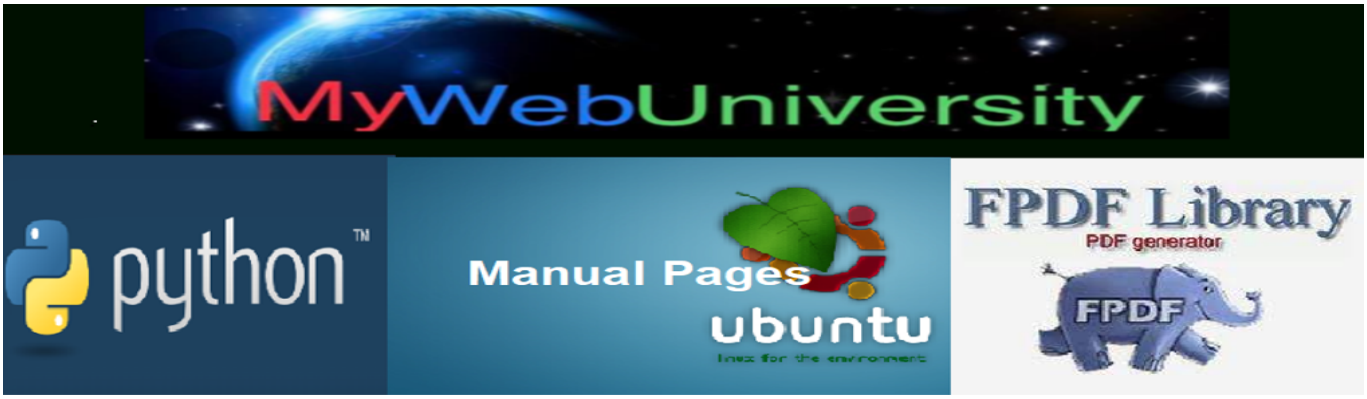

Rocky Enterprise Linux 9.2 Manual Pages on command 'Apache::Session.3pm'

$ man Apache::Session.3pm

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

NAME

Apache::Session - A persistence framework for session data

SYNOPSIS

```
use Apache::Session::MySQL;
my %session;
#make a fresh session for a first-time visitor
tie %session, 'Apache::Session::MySQL';
#stick some stuff in it
$session{visa_number} = "1234 5678 9876 5432";
#get the session id for later use
my $id = $session{_session_id};
#...time passes...
#get the session data back out again during some other request
my %session;
tie %session, 'Apache::Session::MySQL', $id;
validate($session{visa_number});
#delete a session from the object store permanently
tied(%session)->delete;
```

DESCRIPTION

Apache::Session is a persistence framework which is particularly useful for tracking session data between httpd requests. Apache::Session is designed to work with Apache and mod_perl, but it should work under CGI and other web servers, and it also works outside of

a web server altogether.

Apache::Session consists of five components: the interface, the object store, the lock manager, the ID generator, and the serializer. The interface is defined in Session.pm, which is meant to be easily subclassed. The object store can be the filesystem, a Berkeley DB, a MySQL DB, an Oracle DB, a Postgres DB, Sybase, or Informix. Locking is done by lock files, semaphores, or the locking capabilities of the various databases. Serialization is done via Storable, and optionally ASCII-fied via MIME or pack(). ID numbers are generated via MD5. The reader is encouraged to extend these capabilities to meet his own requirements.

A derived class of Apache::Session is used to tie together the three following components. The derived class inherits the interface from Apache::Session, and specifies which store and locker classes to use. Apache::Session::MySQL, for instance, uses the MySQL storage class and also the MySQL locking class. You can easily plug in your own object store or locker class.

INTERFACE

The interface to Apache::Session is very simple: tie a hash to the desired class and use the hash as normal. The constructor takes two optional arguments. The first argument is the desired session ID number, or undef for a new session. The second argument is a hash of options that will be passed to the object store and locker classes.

tieing the session

Get a new session using DBI:

```
tie %session, 'Apache::Session::MySQL', undef,
  { DataSource => 'dbi:mysql:sessions' };
```

Restore an old session from the database:

```
tie %session, 'Apache::Session::MySQL', $session_id,
  { DataSource => 'dbi:mysql:sessions' };
```

Storing and retrieving data to and from the session

Hey, how much easier could it get?

```
$session{first_name} = "Chuck";
$session{an_array_ref} = [ $one, $two, $three ];
$session{an_object} = Some::Class->new;
```

Reading the session ID

The session ID is the only magic entry in the session object, but anything beginning with

an "_" is considered reserved for future use.

```
my $id = $session{_session_id};
```

Permanently removing the session from storage

```
tied(%session)->delete;
```

BEHAVIOR

Apache::Session tries to behave the way the author believes that you would expect. When you create a new session, Session immediately saves the session to the data store, or calls die() if it cannot. It also obtains an exclusive lock on the session object. If you retrieve an existing session, Session immediately restores the object from storage, or calls die() in case of an error. Session also obtains a non-exclusive lock on the session.

As you put data into the session hash, Session squirrels it away for later use. When you untie() the session hash, or it passes out of scope, Session checks to see if anything has changed. If so, Session gains an exclusive lock and writes the session to the data store. It then releases any locks it has acquired.

Note that Apache::Session does only a shallow check to see if anything has changed. If nothing changes in the top level tied hash, the data will not be updated in the backing store. You are encouraged to timestamp the session hash so that it is sure to be updated.

When you call the delete() method on the session object, the object is immediately removed from the object store, if possible.

When Session encounters an error, it calls die(). You will probably want to wrap your session logic in an eval block to trap these errors.

LOCKING AND TRANSACTIONS

By default, most Apache::Session implementations only do locking to prevent data corruption. The locking scheme does not provide transactional consistency, such as you might get from a relational database. If you desire transactional consistency, you must provide the Transaction argument with a true value when you tie the session hash. For example:

```
tie %s, 'Apache::Session::File', $id {
   Directory     => '/tmp/sessions',
   LockDirectory => '/var/lock/sessions',
   Transaction   => 1
};
```

Note that the Transaction argument has no practical effect on the MySQL and Postgres implementations. The MySQL implementation only supports exclusive locking, and the Postgres implementation uses the transaction features of that database.

IMPLEMENTATION

The way you implement Apache::Session depends on what you are trying to accomplish. Here are some hints on which classes to use in what situations

STRATEGIES

Apache::Session is mainly designed to track user session between http requests. However, it can also be used for any situation where data persistence is desirable. For example, it could be used to share global data between your httpd processes. The following examples are short mod_perl programs which demonstrate some session handling basics.

Sharing data between Apache processes

When you share data between Apache processes, you need to decide on a session ID number ahead of time and make sure that an object with that ID number is in your object store before starting your Apache. How you accomplish that is your own business. I use the session ID "1". Here is a short program in which we use Apache::Session to store out database access information.

```
use Apache;
use Apache::Session::File;
use DBI;
use strict;
my %global_data;
eval {
    tie %global_data, 'Apache::Session::File', 1,
      {Directory => '/tmp/sessiondata'};
};
if ($@) {
   die "Global data is not accessible: $@";
}
my $dbh = DBI->connect($global_data{datasource},
   $global_data{username}, $global_data{password}) || die $DBI::errstr;
undef %global_data;
#program continues...
```

As shown in this example, you should undef or untie your session hash as soon as you are done with it. This will free up any locks associated with your process.

Tracking users with cookies

The choice of whether to use cookies or path info to track user IDs is a rather religious topic among Apache users. This example uses cookies. The implementation of a path info system is left as an exercise for the reader.

Note that Apache::Session::Generate::ModUsertrack uses Apache's mod_usertrack cookies to generate and maintain session IDs.

```
use Apache::Session::MySQL;
use Apache;
use strict;
#read in the cookie if this is an old session
my $r = Apache->request;
my $cookie = $r->header_in('Cookie');
$cookie =~ s/SESSION_ID=(\w*)/$1/;
#create a session object based on the cookie we got from the browser,
#or a new session if we got no cookie
my %session;
tie %session, 'Apache::Session::MySQL', $cookie, {
    DataSource => 'dbi:mysql:sessions', #these arguments are
    UserName   => 'mySQL_user',         #required when using
    Password   => 'password',           #MySQL.pm
    LockDataSource => 'dbi:mysql:sessions',
    LockUserName   => 'mySQL_user',
    LockPassword   => 'password'
};
#Might be a new session, so lets give them their cookie back
my $session_cookie = "SESSION_ID=$session{_session_id};";
$r->header_out("Set-Cookie" => $session_cookie);
#program continues...
```

SEE ALSO

Apache::Session::MySQL, Apache::Session::Postgres, Apache::Session::File, Apache::Session::DB_File, Apache::Session::Oracle, Apache::Session::Sybase

The O Reilly book "Apache Modules in Perl and C", by Doug MacEachern and Lincoln Stein, has a chapter on keeping state.

CGI::Session uses OO interface to do same thing. It is better maintained, but less possibilies.

Catalyst::Plugin::Session - support of sessions in Catalyst

Session - OO interface to Apache::Session

LICENSE

Under the same terms as Perl itself.

AUTHORS

Alexandr Ciornii, <http://chorny.net> - current maintainer

Jeffrey Baker <jwbaker@acm.org> is the author of Apache::Session.

Tatsuhiko Miyagawa <miyagawa@bulknews.net> is the author of Generate::ModUniqueID and Generate::ModUsertrack

Erik Rantapaa <rantapaa@fanbuzz.com> found errors in both Lock::File and Store::File

Bart Schaefer <schaefer@zanshin.com> notified me of a bug in Lock::File.

Chris Winters <cwinters@intes.net> contributed the Sybase code.

Michael Schout <mschout@gkg.net> fixed a commit policy bug in 1.51.

Andreas J. Koenig <andreas.koenig@anima.de> contributed valuable CPAN advice and also Apache::Session::Tree and Apache::Session::Counted.

Gerald Richter <richter@ecos.de> had the idea for a tied hash interface and provided the initial code for it. He also uses Apache::Session in his Embperl module and is the author of Apache::Session::Embperl

Jochen Wiedmann <joe@ipsoft.de> contributed patches for bugs and improved performance.

Steve Shreeve <shreeve@uci.edu> squashed a bug in 0.99.0 whereby a cleared hash or deleted key failed to set the modified bit.

Peter Kaas <Peter.Kaas@lunatech.com> sent quite a bit of feedback with ideas for interface improvements.

Randy Harmon <rjharmon@uptimecomputers.com> contributed the original storage-independent object interface with input from:

Bavo De Ridder <bavo@ace.ulyssis.student.kuleuven.ac.be>

Jules Bean <jmlb2@hermes.cam.ac.uk>

Lincoln Stein <lstein@cshl.org>

Jamie LeTaul <jletual@kmtechnologies.com> fixed file locking on Windows.

Scott McWhirter <scott@surreytech.co.uk> contributed verbose error messages for file locking.

Corris Randall <corris@line6.net> gave us the option to use any table name in the MySQL store.

Oliver Maul <oliver.maul@ixos.de> updated the Sybase modules

Innumerable users sent a patch for the reversed file age test in the file locking module.

Langen Mike <mike.langen@tamedia.ch> contributed Informix modules.