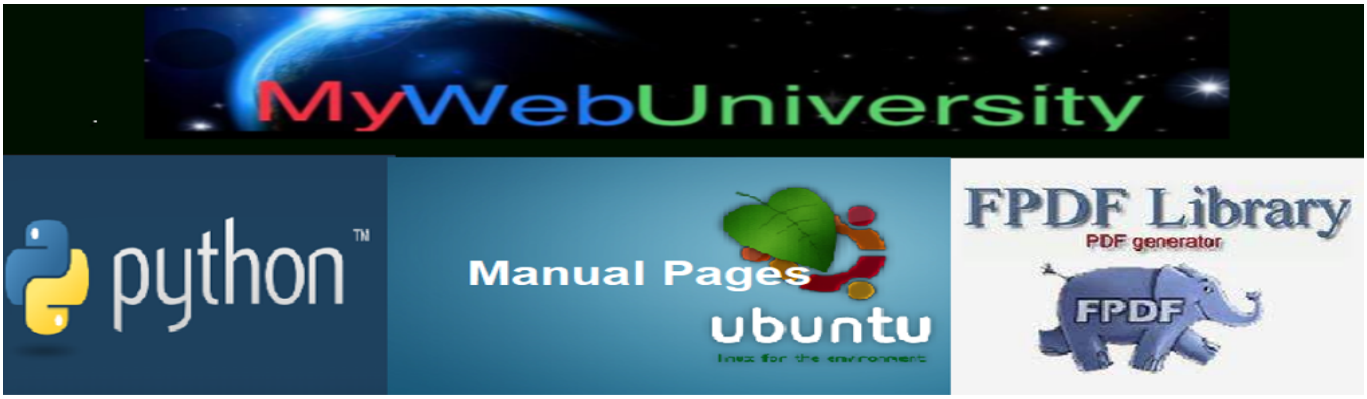

Rocky Enterprise Linux 9.2 Manual Pages on command 'Apache2::ServerUtil.3pm'

$ man Apache2::ServerUtil.3pm

libapache2-mod-perl2-2.0.12::doUslibapache2-mod-perl2-2.0.12::docs::api::Apache2::ServerUtil(3pm)

NAME

Apache2::ServerUtil - Perl API for Apache server record utils

Synopsis

```
use Apache2::ServerUtil ();
$s = Apache2::ServerUtil->server;
# push config
$s->add_config(['ServerTokens off']);
# add components to the Server signature
$s->add_version_component("MyModule/1.234");
# access PerlSetVar/PerlAddVar values
my $srv_cfg = $s->dir_config;
# check command line defines
print "this is mp2"
   if Apache2::ServerUtil::exists_config_define('MODPERL2');
# get PerlChildExitHandler configured handlers
@handlers = @{ $s->get_handlers('PerlChildExitHandler') || []};
# server build and version info:
$when_built = Apache2::ServerUtil::get_server_built();
$description = Apache2::ServerUtil::get_server_description();
$version = Apache2::ServerUtil::get_server_version();
$banner = Apache2::ServerUtil::get_server_banner();
# ServerRoot value
```

```
    $server_root = Apache2::ServerUtil::server_root();
    # get 'conf/' dir path (avoid using this function!)
    my $dir = Apache2::ServerUtil::server_root_relative($r->pool, 'conf');
    # set child_exit handlers
    $r->set_handlers(PerlChildExitHandler => \&handler);
    # server level PerlOptions flags lookup
    $s->push_handlers(ChildExit => \&child_exit)
        if $s->is_perl_option_enabled('ChildExit');
    # extend HTTP to support a new method
    $s->method_register('NEWGET');
    # register server shutdown callback
    Apache2::ServerUtil::server_shutdown_register_cleanup(sub { Apache2::Const::OK });
    # do something only when the server restarts
    my $cnt = Apache2::ServerUtil::restart_count();
    do_something_once() if $cnt > 1;
    # get the resolved ids from Group and User entries
    my $user_id  = Apache2::ServerUtil->user_id;
    my $group_id = Apache2::ServerUtil->group_id;
```

Description

"Apache2::ServerUtil" provides the Apache server object utilities API.

Methods API

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

"add_config"

Dynamically add Apache configuration:

```
  $s->add_config($lines);
```

obj: $s ("Apache2::ServerRec object")

arg1: $lines (ARRAY ref)

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

ret: no return value

since: 2.0.00

See also: "$r->add_config"

For example:

Add a configuration section at the server startup (e.g. from startup.pl):

```
use Apache2::ServerUtil ();
my $conf = <<'EOC';
PerlModule Apache2::MyExample
<Location /perl>
  SetHandler perl-script
  PerlResponseHandler Apache2::MyExample
</Location>
EOC
Apache2::ServerUtil->server->add_config([split /\n/, $conf]);
```

"add_version_component"

Add a component to the version string

```
$s->add_version_component($component);
```

obj: $s ("Apache2::ServerRec object")

arg1: $component (string)

The string component to add

ret: no return value

since: 2.0.00

This function is usually used by modules to advertise themselves to the world. It's picked up by such statistics collectors, like netcraft.com, which accomplish that by connecting to various servers and grabbing the server version response header ("Server"). Some servers choose to fully or partially conceal that header.

This method should be invoked in the "PerlPostConfigHandler" phase, which will ensure that the Apache core version number will appear first.

For example let's add a component "Hikers, Inc/0.99999" to the server string at the server startup:

```
use Apache2::ServerUtil ();
use Apache2::Const -compile => 'OK';
Apache2::ServerUtil->server->push_handlers(
    PerlPostConfigHandler => \&add_my_version);
sub add_my_version {
    my ($conf_pool, $log_pool, $temp_pool, $s) = @_;
    $s->add_version_component("Hikers, Inc/0.99999");
```

```
    return Apache2::Const::OK;
}
```

or of course you could register the "PerlPostConfigHandler" handler directly in httpd.conf

Now when the server starts, you will something like:

```
 [Thu Jul 15 12:15:28 2004] [notice] Apache/2.0.51-dev (Unix)
 mod_perl/1.99_15-dev Perl/v5.8.5 Hikers, Inc/0.99999
 configured -- resuming normal operations
```

Also remember that the "ServerTokens" directive value controls whether the component information is displayed or not.

"dir_config"

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

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

obj: $s ("Apache2::ServerRec 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.

```
 $t = $s->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 APR::Table.

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

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

```
 $value = $s->dir_config($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.

```
$s->dir_config($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.

```
$s->dir_config($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.

"exists_config_define"

Check for a definition from the server startup command line (e.g. "-DMODPERL2")

```
$result = Apache2::ServerUtil::exists_config_define($name);
```

arg1: $name (string)

The define string to check for

ret: $result (boolean)

true if defined, false otherwise

since: 2.0.00

For example:

```
print "this is mp2"
  if Apache2::ServerUtil::exists_config_define('MODPERL2');
```

"get_handlers"

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

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

obj: $s ("Apache2::ServerRec object")

arg1: $hook_name (string)

a string representing the phase to handle.

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

a list of references to the handler subroutines

since: 2.0.00

See also: "$r->add_config"

For example:

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

@handlers = @{ $s->get_handlers('PerlChildExitHandler') || []};

"get_server_built"

Get the date and time that the server was built

$when_built = Apache2::ServerUtil::get_server_built();

ret: $when_built (string)

The server build time string

since: 2.0.00

"get_server_version"

Get the server version string

$version = Apache2::ServerUtil::get_server_version();

ret: $version (string)

The server version string

since: 2.0.00

"get_server_banner"

Get the server banner

$banner = Apache2::ServerUtil::get_server_banner();

ret: $banner (string)

The server banner

since: 2.0.4

"get_server_description"

Get the server description

$description = Apache2::ServerUtil::get_server_description();

ret: $description (string)

The server description

since: 2.0.4

"group_id"

Get the group id corresponding to the "Group" directive in httpd.conf:

$gid = Apache2::ServerUtil->group_id;

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

ret: $gid (integer)

On Unix platforms returns the gid corresponding to the value used in the "Group"

directive in httpd.conf. On other platforms returns 0.

since: 2.0.03

"is_perl_option_enabled"

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

$result = $s->is_perl_option_enabled($flag);

obj: $s ("Apache2::ServerRec object")

arg1: $flag (string)

ret: $result (boolean)

since: 2.0.00

For example to check whether the "ChildExit" hook is enabled (which can be disabled with "PerlOptions -ChildExit") and configure some handlers to run if enabled:

```
$s->push_handlers(ChildExit => \&child_exit)
    if $s->is_perl_option_enabled('ChildExit');
```

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

"method_register"

Register a new request method, and return the offset that will be associated with that method.

$offset = $s->method_register($methname);

obj: $s ("Apache2::ServerRec object")

arg1: $methname (string)

The name of the new method to register (in addition to the already supported "GET", "HEAD", etc.)

ret: $offset (integer)

An int value representing an offset into a bitmask. You can probably ignore it.

since: 2.0.00

This method allows you to extend the HTTP protocol to support new methods, which fit the HTTP paradigm. Of course you will need to write a client that understands that protocol extension. For a good example, refer to the "MyApache2::SendEmail" example presented in "the PerlHeaderParserHandler section", which demonstrates how a new method "EMAIL" is registered and used.

"push_handlers"

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

$ok = $s->push_handlers($hook_name => \&handler);

```
  $ok = $s->push_handlers($hook_name => [\&handler, \&handler2]);
```

obj: $s ("Apache2::ServerRec 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: "$r->add_config"

Examples:

A single handler:

```
  $s->push_handlers(PerlChildExitHandler => \&handler);
```

Multiple handlers:

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

Anonymous functions:

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

"restart_count"

How many times the server was restarted.

```
  $restart_count = Apache2::ServerUtil::restart_count();
```

ret: "restart_count" (number)

since: 2.0.00

The following demonstration should make it clear what values to expect from this function.

Let's add the following code to startup.pl, so it's run every time httpd.conf is parsed:

```
  use Apache2::ServerUtil ();
  my $cnt = Apache2::ServerUtil::restart_count();
  open my $fh, ">>/tmp/out" or die "$!";
  print $fh "cnt: $cnt\n";
  close $fh;
```

Now let's run a series of server starts and restarts and look at what is logged into

/tmp/out:

```
% httpd -k start
cnt: 1
cnt: 2
% httpd -k graceful
cnt: 1
cnt: 3
% httpd -k graceful
cnt: 1
cnt: 4
% httpd -k stop
cnt: 1
```

Remembering that Apache restarts itself immediately after starting, we can see that the "restart_count" goes from 1 to 2 during the server start. Moreover we can see that every operation forces the parsing of httpd.conf and therefore reinitialization of mod_perl (and running all the code found in httpd.conf). This happens even when the server is shutdown via "httpd -k stop".

What conclusions can be drawn from this demonstration:

? "Apache2::ServerUtil::restart_count()" returns 1 every time some "-k" command is passed to Apache (or "kill -USR1" or some alternative signal is received).

? At all other times the count will be 2 or higher. So for example on graceful restart the count will be 3 or higher.

For example if you want to run something every time "httpd -k" is run you just need to check whether "restart_count()" returns 1:

```
my $cnt = Apache2::ServerUtil::restart_count();
do_something() if $cnt == 1;
```

To do something only when server restarts ("httpd -k start" or "httpd -k graceful)", check whether "restart_count()" is bigger than 1:

```
my $cnt = Apache2::ServerUtil::restart_count();
do_something() if $cnt > 1;
```

"server"

Get the main server's object

```
$main_s = Apache2::ServerUtil->server();
```

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

ret: $main_s ("Apache2::ServerRec object")

since: 2.0.00

"server_root"

returns the value set by the top-level "ServerRoot" directive.

```
$server_root = Apache2::ServerUtil::server_root();
```

ret: $server_root (string)

since: 2.0.00

"server_root_relative"

Returns the canonical form of the filename made absolute to "ServerRoot":

```
$path = Apache2::ServerUtil::server_root_relative($pool, $fname);
```

arg1: $pool ("APR::Pool object")

Make sure that you read the following explanation and understand well which pool object you need to pass before using this function.

opt arg2: $fname (string)

ret: $path (string)

The concatenation of "ServerRoot" and the $fname.

If $fname is not specified, the value of "ServerRoot" is returned with a trailing "/".

(it's the same as using '' as $fname's value).

since: 2.0.00

$fname is appended to the value of "ServerRoot" and returned. For example:

```
my $dir = Apache2::ServerUtil::server_root_relative($r->pool, 'logs');
```

You must be extra-careful when using this function. If you aren't sure what you are doing don't use it.

It's much safer to build the path by yourself using use "Apache2::ServerUtil::server_root()", For example:

```
use File::Spec::Functions qw(catfile);
my $path = catfile Apache2::ServerUtil::server_root, qw(t logs);
```

In this example, no memory allocation happens on the Apache-side and you aren't risking to get a memory leak.

The problem with "server_root_relative" is that Apache allocates memory to concatenate the path string. The memory is allocated from the pool object. If you call this method on the server pool object it'll allocate the memory from it. If you do that at the server

startup, it's perfectly right, since you will do that only once. However if you do that from within a request or a connection handler, you create a memory leak every time it is called -- as the memory gets allocated from the server pool, it will be freed only when the server is shutdown. Therefore if you need to build a relative to the root server path for the duration of the request, use the request pool:

```
use Apache2::RequestRec ();
Apache2::ServerUtil::server_root_relative($r->pool, $fname);
```

If you need to have the path for the duration of a connection (e.g. inside a protocol handler), you should use:

```
use Apache2::Connection ();
Apache2::ServerUtil::server_root_relative($c->pool, $fname);
```

And if you want it for the scope of the server file:

```
use Apache2::Process ();
use Apache2::ServerUtil ();
Apache2::ServerUtil::server_root_relative($s->process->pool, $fname);
```

Moreover, you could have encountered the opposite problem, where you have used a short-lived pool object to construct the path, but tried to use the resulting path variable, when that pool has been destructed already. In order to avoid mysterious segmentation faults, mod_perl does a wasteful copy of the path string when returning it to you -- another reason to avoid using this function.

"server_shutdown_cleanup_register"

Register server shutdown cleanup callback:

```
Apache2::ServerUtil::server_shutdown_cleanup_register($sub);
```

arg1: $sub (CODE ref or SUB name)

ret: no return value

since: 2.0.00

This function can be used to register a callback to be run once at the server shutdown (compared to "PerlChildExitHandler" which will execute the callback for each exiting child process).

For example in order to arrange the function "do_my_cleanups()" to be run every time the server shuts down (or restarts), run the following code at the server startup:

```
Apache2::ServerUtil::server_shutdown_cleanup_register(\&do_my_cleanups);
```

It's necessary to run this code at the server startup (normally startup.pl). The function

will croak if run after the "PerlPostConfigHandler" phase.

Values returned from cleanup functions are ignored. If a cleanup dies the exception is stringified and passed to "warn()". Usually, this results in printing it to the error_log.

"set_handlers"

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

```
$ok = $s->set_handlers($hook_name => \&handler);
$ok = $s->set_handlers($hook_name => [\&handler, \&handler2]);
$ok = $s->set_handlers($hook_name => []);
$ok = $s->set_handlers($hook_name => undef);
```

obj: $s ("Apache2::ServerRec 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: "$r->add_config"

Examples:

A single handler:

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

"user_id"

Get the user id corresponding to the "User" directive in httpd.conf:

```
$uid = Apache2::ServerUtil->user_id;
```

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

ret: $uid (integer)

On Unix platforms returns the uid corresponding to the value used in the "User" directive in httpd.conf. On other platforms returns 0.

since: 2.0.03

Unsupported API

"Apache2::ServerUtil" also provides auto-generated Perl interface for a few other methods which aren't tested at the moment and therefore their API is a subject to change. These methods will be finalized later as a need arises. If you want to rely on any of the following methods please contact the the mod_perl development mailing list so we can help each other take the steps necessary to shift the method to an officially supported API.

"error_log2stderr"

Start sending STDERR to the error_log file

```
$s->error_log2stderr();
```

obj: $s ("Apache2::ServerRec object")

The current server

ret: no return value

since: 2.0.00

This method may prove useful if you want to start redirecting STDERR to the error_log file before Apache does that on the startup.

See Also

mod_perl 2.0 documentation.

Copyright

Authors

The mod_perl development team and numerous contributors.