1 Apache2::ServerRec - Perl API for Apache server record accessors

1.1 Synopsis

```
use Apache2::ServerRec ();

$error_fname = $s->error_fname();

$is_virtual = $s->is_virtual();

$keep_alive         = $s->keep_alive();
$keep_alive_max     = $s->keep_alive_max();
$keep_alive_timeout = $s->keep_alive_timeout();

$limit_req_fields    = $s->limit_req_fields();
$limit_req_fieldsize = $s->limit_req_fieldsize();
$limit_req_line      = $s->limit_req_line();

$path = $s->path();

$hostname = $s->server_hostname();
$port     = $s->port();

$server_admin = $s->server_admin();

$proc = $s->process();

$timeout  = $s->timeout();
$loglevel = $s->loglevel();

my $server = Apache2::ServerUtil->server;
my $vhosts = 0;
for (my $s = $server->next; $s; $s = $s->next) {
    $vhosts++;
}
print "There are $vhosts virtual hosts";
```

1.2 Description

`Apache2::ServerRec` provides the Perl API for Apache server_rec object.

`Apache2::ServerUtil` provides an extra functionality.

1.3 API

`Apache2::ServerRec` provides the following functions and/or methods:

1.3.1 *`error_fname`*

Get/set the `ErrorLog` file value (e.g. *logs/error_log*)

```
  $error_fname      = $s->error_fname();
  $prev_error_fname = $s->error_fname($new_error_fname);
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **opt arg1: `$new_error_fname` (string)**

  If passed, sets the new value for `ErrorLog`

  Note the limited functionality under threaded MPMs.

- **ret: `$error_fname` (string)**

  Returns the `ErrorLog` value setting.

  If `$new_error_fname` is passed returns the setting before the change.

- **since: 2.0.00**

1.3.2 `is_virtual`

Test whether `$s` is a virtual host object

```
  $is_virtual = $s->is_virtual();
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **ret: `$is_virtual` (boolean)**

  Returns the is_virtual setting.

  If `$new_is_virtual` is passed, returns the setting before the change.

- **since: 2.0.00**

Example:

```
  print "This is a virtual host" if $s->is_virtual();
```

1.3.3 `keep_alive`

Get/set the `KeepAlive` setting, which specifies whether Apache should accept more than one request over the same connection from the same client.

```
  $keep_alive      = $s->keep_alive();
  $prev_keep_alive = $s->keep_alive($new_keep_alive);
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **opt arg1: `$new_keep_alive` (boolean)**

If passed, sets the new keep_alive.

Note the limited functionality under threaded MPMs.

- **ret: `$keep_alive` (boolean)**

  Returns the `KeepAlive` setting.

  If `$new_keep_alive` is passed, returns the setting before the change.

- **since: 2.0.00**

1.3.4 `keep_alive_max`

Get/set the `MaxKeepAliveRequest` setting, which specifies the maximum number of requests Apache will serve over a `KeepAlive` connection.

```
$keep_alive_max      = $s->keep_alive_max();
$prev_keep_alive_max = $s->keep_alive_max($new_keep_alive_max);
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **opt arg1: `$new_keep_alive_max` (integer)**

  If passed, sets the new keep_alive_max.

  Note the limited functionality under threaded MPMs.

- **ret: `$keep_alive_max` (integer)**

  Returns the keep_alive_max setting.

  If `$new_keep_alive_max` is passed, returns the setting before the change.

- **since: 2.0.00**

1.3.5 `keep_alive_timeout`

Get/set the `KeepAliveTimeout` setting (in microsecs), which specifies how long Apache will wait for another request before breaking a `KeepAlive` connection.

```
$keep_alive_timeout      = $s->keep_alive_timeout();
$prev_keep_alive_timeout = $s->keep_alive_timeout($new_timeout);
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **opt arg1: `$new_keep_alive_timeout` (integer)**

  The expected value is in microsecs.

If passed, sets the new `KeepAlive` timeout.

Note the limited functionality under threaded MPMs.

- **ret: `$keep_alive_timeout` (integer)**

  Returns the `KeepAlive` timeout value (in microsecs).

  If `$new_timeout` is passed, returns the setting before the change.

- **since: 2.0.00**

1.3.6 *`limit_req_fields`*

Get/set limit on number of request header fields

```
$limit_req_fields      = $s->limit_req_fields();
$prev_limit_req_fields = $s->limit_req_fields($new_limit_req_fields);
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **opt arg1: `$new_limit_req_fields` (integer)**

  If passed, sets the new request headers number limit.

  Note the limited functionality under threaded MPMs.

- **ret: `$limit_req_fields` (integer)**

  Returns the request headers number limit.

  If `$new_limit_req_fields` is passed, returns the setting before the change.

- **since: 2.0.00**

1.3.7 *`limit_req_fieldsize`*

Get/set limit on size of any request header field

```
$limit_req_fieldsize = $s->limit_req_fieldsize();
$prev_limit          = $s->limit_req_fieldsize($new_limit);
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **opt arg1: `$new_limit_req_fieldsize` (integer)**

  If passed, sets the new request header size limit.

  Note the limited functionality under threaded MPMs.

- **ret: `$limit_req_fieldsize` (integer)**

  Returns the request header size limit.

  If `$new_limit` is passed, returns the setting before the change.

- **since: 2.0.00**

1.3.8 *`limit_req_line`*

Get/set limit on size of the HTTP request line

```
$limit_req_line      = $s->limit_req_line();
$prev_limit_req_line = $s->limit_req_line($new_limit_req_line);
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **opt arg1: `$new_limit_req_line` (integer)**

  If passed, sets the new request line limit value.

  Note the limited functionality under threaded MPMs.

- **ret: `$limit_req_line` (integer)**

  Returns the request line limit value

  If `$new_limit_req_line` is passed, returns the setting before the change.

- **since: 2.0.00**

1.3.9 *`loglevel`*

Get/set the `LogLevel` directive value

```
$loglevel      = $s->loglevel();
$prev_loglevel = $s->loglevel($new_loglevel);
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **opt arg1: `$new_loglevel` (`Apache2::Const :log constant`)**

  If passed, sets a new `LogLevel` value

  Note the limited functionality under threaded MPMs.

- **ret: `$loglevel` (`Apache2::Const :log constant`)**

  Returns the `LogLevel` value as a constant.

If `$new_loglevel` is passed, returns the setting before the change.

- **since: 2.0.00**

For example, to set the `LogLevel` value to `info`:

```
use Apache2::Const -compile => qw(LOG_INFO);
$s->loglevel(Apache2::Const::LOG_INFO);
```

1.3.10 `next`

The next server record in the list (if there are vhosts)

```
$s_next = $s->next();
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **ret: `$s_next` (`Apache2::ServerRec object`)**
- **since: 2.0.00**

For example the following code traverses all the servers, starting from the base server and continuing to vhost servers, counting all available vhosts:

```
use Apache2::ServerRec ();
use Apache2::ServerUtil ();
my $server = Apache2::ServerUtil->server;
my $vhosts = 0;
for (my $s = $server->next; $s; $s = $s->next) {
    $vhosts++;
}
print "There are $vhosts virtual hosts";
```

1.3.11 `path`

Get/set pathname for the `ServerPath` setting

```
$path      = $s->path();
$prev_path = $s->path($new_path);
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **opt arg1: `$new_path` (string)**

  If passed, sets the new path.

  Note the limited functionality under threaded MPMs.

- **ret: `$path` (string)**

  Returns the path setting.

If `$new_path` is passed, returns the setting before the change.

- **since: 2.0.00**

1.3.12 *`port`*

Get/set the port value

```
$port      = $s->port();
$prev_port = $s->port($new_port);
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **opt arg1: `$new_port` (integer)**

  If passed, sets the new port.

  Note the limited functionality under threaded MPMs.

  META: I don’t think one should be allowed to change port number after the server has started.

- **ret: `$port` (integer)**

  Returns the port setting.

  If `$new_port` is passed returns the setting before the change.

- **since: 2.0.00**

1.3.13 *`process`*

The process this server is running in

```
$proc = $s->process();
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **ret: `$proc` (`Apache2::Process object`)**
- **since: 2.0.00**

1.3.14 *`server_admin`*

Get/set the `ServerAdmin` value

```
$server_admin      = $s->server_admin();
$prev_server_admin = $s->server_admin($new_server_admin);
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **opt arg1: `$new_server_admin` (string)**

If passed, sets the new `ServerAdmin` value.

Note the limited functionality under threaded MPMs.

- **ret: `$server_admin` (string)**

  Returns the `ServerAdmin` value.

  If `$new_server_admin` is passed, returns the setting before the change.

- **since: 2.0.00**

1.3.15 *`server_hostname`*

Get/set the `ServerName` value

```
$server_hostname      = $s->server_hostname();
$prev_server_hostname = $s->server_hostname($new_server_hostname);
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **opt arg1: `$new_server_hostname` (string)**

  If passed, sets the `ServerName` value

  Note the limited functionality under threaded MPMs.

- **ret: `$server_hostname` (string)**

  Returns the `ServerName` value

  If `$new_server_hostname` is passed, returns the setting before the change.

- **since: 2.0.00**

1.3.16 *`timeout`*

Get/set the timeout (`TimeOut`) (in microsecs), which Apache will wait for before it gives up doing something

```
$timeout      = $s->timeout();
$prev_timeout = $s->timeout($new_timeout);
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **opt arg1: `$new_timeout` (integer)**

  If passed, sets the new timeout (the value should be in microseconds).

Note the limited functionality under threaded MPMs.

- **ret: `$timeout` (integer)**

  Returns the timeout setting in microseconds.

  If `$new_timeout` is passed, returns the setting before the change.

- **since: 2.0.00**

Let us repeat again: the timeout values is microseconds. For example to set the timeout to 20 secs:

```
$s->timeout(20_000_000);
```

1.4 Notes

1.4.1 Limited Functionality under Threaded MPMs

Note that under threaded MPMs, some of the read/write accessors, will be able to set values only before threads are spawned (i.e. before the `ChildInit phase`). Therefore if you are developing your application on the non-threaded MPM, but planning to have it run under threaded mpm, you should not use those methods to set values after the ChildInit phase.

The affected accessor methods are marked as such in their respective documentation entries.

1.5 Unsupported API

`Apache2::ServerRec` 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.

1.5.1 `addrs`

Get the addrs value

```
$addrs = $s->addrs();
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **ret: `$addrs` (*`Apache2::ServerAddr`*)**

  Returns the addrs setting.

- **since: subject to change**

META: this methods returns a vhost-specific Apache2::ServerAddr object, which is not implemented at the moment. See the struct server_addr_rec entry in httpd-2.0/include/httpd.h for more information. It seems that most (all?) of the information in that record is available through other APIs.

1.5.2 *`lookup_defaults`*

Get the lookup_defaults value. MIME type info, etc., before we start checking per-directory info.

```
$lookup_defaults = $s->lookup_defaults();
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **ret: `$lookup_defaults` (`Apache2::ConfVector`)**

  Returns the lookup_defaults setting.

- **since: subject to change**

1.5.3 *`module_config`*

Get config vector containing pointers to modules’ per-server config structures.

```
$module_config = $s->module_config();
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **ret: `$module_config` (`Apache2::ConfVector`)**

  Returns the module_config setting.

- **since: subject to change**

1.5.4 *`names`*

Get/set the value(s) for the `ServerAlias` setting

```
$names      = $s->names();
$prev_names = $s->names($new_names);
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **opt arg1: `$new_names` (*`APR::ArrayHeader`*)**

  If passed, sets the new names.

  Note the limited functionality under threaded MPMs.

- **ret: `$names` (*`APR::ArrayHeader`*)**

  Returns the names setting.

If `$new_names` is passed, returns the setting before the change.

- **since: 2.0.00**

META: we don't have `APR::ArrayHeader` yet

1.5.5 *`wild_names`*

Wildcarded names for ServerAlias servers

```
$wild_names      = $s->wild_names();
$prev_wild_names = $s->wild_names($new_wild_names);
```

- **obj: `$s` (`Apache2::ServerRec object`)**
- **opt arg1: `$new_wild_names` (*`APR::ArrayHeader`*)**

  If passed, sets the new wild_names.

  Note the limited functionality under threaded MPMs.

- **ret: `$wild_names` (*`APR::ArrayHeader`*)**

  Returns the wild_names setting.

  If `$new_wild_names` is passed, returns the setting before the change.

- **since: 2.0.00**

META: we don't have `APR::ArrayHeader` yet

1.6 See Also

mod_perl 2.0 documentation.

1.7 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 2.0.

1.8 Authors

The mod_perl development team and numerous contributors.

Table of Contents: