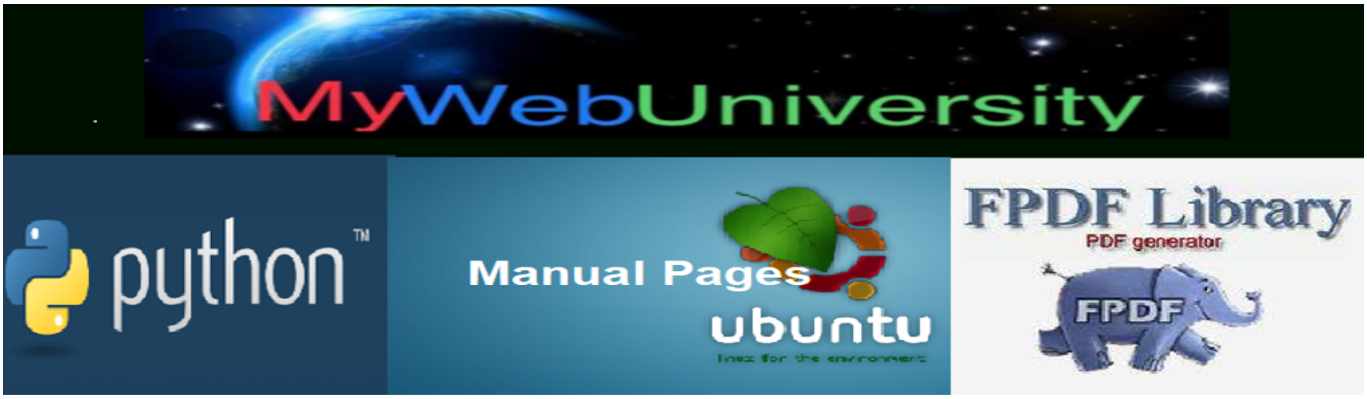

Rocky Enterprise Linux 9.2 Manual Pages on command 'Apache2::Connection.3pm'

$ man Apache2::Connection.3pm

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

NAME

Apache2::Connection - Perl API for Apache connection object

Synopsis

```
use Apache2::Connection ();
use Apache2::RequestRec ();
my $c = $r->connection;
my $c = $r->connection;
# is connection still open?
$status = $c->aborted;
# base server
$base_server = $c->base_server();
# needed for creating buckets/brigades
$ba = $c->bucket_alloc();
# client's socket
$socket = $c->client_socket;
# unique connection id
$id = $c->id();
# connection filters stack
$input_filters = $c->input_filters();
$output_filters = $c->output_filters();
# keep the connection alive?
$status = $c->keepalive();
```

```
    # how many requests served over the current connection
    $served = $c->keepalives();
    # this connection's local and remote socket addresses
    $local_sa  = $c->local_addr();
    $remote_sa = $c->remote_addr();
    # local and remote hostnames
    $local_host = $c->local_host();
    $remote_host = $c->get_remote_host();
    $remote_host = $c->remote_host();
    # server and remote client's IP addresses
    $local_ip = $c->local_ip();
    $remote_ip = $c->remote_ip();
    # connection level Apache notes
    $notes = $c->notes();
    # this connection's pool
    $p = $c->pool();
```

Description

"Apache2::RequestRec" provides the Perl API for Apache connection record object.

API

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

"aborted"

Check whether the connection is still open

```
  $status = $c->aborted();
```

obj: $c ("Apache2::Connection object")

ret: $status (boolean)

true if the connection has been aborted, false if still open

since: 2.0.00

"base_server"

Physical server this connection came in on (main server or vhost):

```
  $base_server = $c->base_server();
```

obj: $c ("Apache2::Connection object")

ret: $base_server ("Apache2::Server object")

since: 2.0.00

"bucket_alloc"

The bucket allocator to use for all bucket/brigade creations

$ba = $c->bucket_alloc();

obj: $c ("Apache2::Connection object")

ret: $ba ("APR::BucketAlloc object")

since: 2.0.00

This object is needed by "APR::Bucket" and "APR::Brigade" methods/functions.

"client_socket"

Get/set the client socket

$socket = $c->client_socket;

$prev_socket = $c->client_socket($new_socket);

obj: $c ("Apache2::Connection object")

opt arg1: $new_socket ("APR::Socket object" object)

If passed a new socket will be set.

ret: $socket ("APR::Socket object" object)

current client socket

if the optional argument $new_socket was passed the previous socket object is

returned.

since: 2.0.00

"get_remote_host"

Lookup the client's DNS hostname or IP address

$remote_host = $c->remote_host();

$remote_host = $c->remote_host($type);

$remote_host = $c->remote_host($type, $dir_config);

obj: $c ("Apache2::Connection object")

The current connection

opt arg1: $type (":remotehost constant")

The type of lookup to perform:

"Apache2::Const::REMOTE_DOUBLE_REV"

will always force a DNS lookup, and also force a double reverse lookup, regardless

of the "HostnameLookups" setting. The result is the (double reverse checked)

hostname, or undef if any of the lookups fail.

"Apache2::Const::REMOTE_HOST"

returns the hostname, or "undef" if the hostname lookup fails. It will force a DNS lookup according to the "HostnameLookups" setting.

"Apache2::Const::REMOTE_NAME"

returns the hostname, or the dotted quad if the hostname lookup fails. It will force a DNS lookup according to the "HostnameLookups" setting.

"Apache2::Const::REMOTE_NOLOOKUP"

is like "Apache2::Const::REMOTE_NAME" except that a DNS lookup is never forced.

Default value is "Apache2::Const::REMOTE_NAME".

opt arg2: $dir_config ("Apache2::ConfVector object")

The directory config vector from the request. It's needed to find the container in which the directive "HostnameLookups" is set. To get one for the current request use "$r->per_dir_config".

By default, "undef" is passed, in which case it's the same as if "HostnameLookups" was set to "Off".

ret: $remote_host (string/undef)

The remote hostname. If the configuration directive HostNameLookups is set to off, this returns the dotted decimal representation of the client's IP address instead.

Might return "undef" if the hostname is not known.

since: 2.0.00

The result of "get_remote_host" call is cached in "$c->remote_host". If the latter is set, "get_remote_host" will return that value immediately, w/o doing any checkups.

"id"

ID of this connection; unique at any point in time

```
$id = $c->id();
```

obj: $c ("Apache2::Connection object")

ret: $id (integer)

since: 2.0.00

"input_filters"

Get/set the first filter in a linked list of protocol level input filters:

```
$input_filters      = $c->input_filters();
$prev_input_filters = $c->input_filters($new_input_filters);
```

obj: $c ("Apache2::Connection object")

opt arg1: $new_input_filters

Set a new value

ret: $input_filters ("Apache2::Filter object")

The first filter in the connection input filters chain.

If $new_input_filters was passed, returns the previous value.

since: 2.0.00

For an example see: Bucket Brigades-based Protocol Module

"keepalive"

This method answers the question: Should the the connection be kept alive for another HTTP request after the current request is completed?

```
$status = $c->keepalive();
$status = $c->keepalive($new_status);
```

obj: $c ("Apache2::Connection object")

opt arg1: $new_status (":conn_keepalive constant")

Normally you should not mess with setting this option when handling the HTTP protocol.

If you do (for example when sending your own headers set with "$r->assbackwards") --

take a look at the ap_set_keepalive() function in

httpd-2.0/modules/http/http_protocol.c.

ret: $status (":conn_keepalive constant")

The method does not return true or false, but one of the states which can be compared against (":conn_keepalive constants").

since: 2.0.00

Unless you set this value yourself when implementing non-HTTP protocols, it's only relevant for HTTP requests.

For example:

```
use Apache2::RequestRec ();
use Apache2::Connection ();
use Apache2::Const -compile => qw(:conn_keepalive);
...
my $c = $r->connection;
if ($c->keepalive == Apache2::Const::CONN_KEEPALIVE) {
    # do something
}
elsif ($c->keepalive == Apache2::Const::CONN_CLOSE) {
```

```
    # do something else
  }
  elsif ($c->keepalive == Apache2::Const::CONN_UNKNOWN) {
    # do yet something else
  }
  else {
    # die "unknown state";
  }
```

Notice that new states could be added later by Apache, so your code should make no assumptions and do things only if the desired state matches.

"keepalives"

How many requests were already served over the current connection.

```
  $served = $c->keepalives();
  $served = $c->keepalives($new_served);
```

obj: $c ("Apache2::Connection object")

opt arg1: $new_served (integer)

Set the number of served requests over the current connection. Normally you won't do that when handling HTTP requests. (But see below a note regarding "$r->assbackwards").

ret: $served (integer)

How many requests were already served over the current connection.

In most handlers, but HTTP output filter handlers, that value doesn't count the current request. For the latter it'll count the current request.

since: 2.0.00

This method is only relevant for keepalive connections. The core connection output filter "ap_http_header_filter" increments this value when the response headers are sent and it decides that the connection should not be closed (see "ap_set_keepalive()").

If you send your own set of HTTP headers with "$r->assbackwards", which includes the "Keep-Alive" HTTP response header, you must make sure to increment the "keepalives" counter.

"local_addr"

Get this connection's local socket address

```
  $local_sa = $c->local_addr();
```

obj: $c ("Apache2::Connection object")

ret: $local_sa ("APR::SockAddr object")

since: 2.0.00

"local_host"

used for ap_get_server_name when UseCanonicalName is set to DNS (ignores setting of HostnameLookups)

```
$local_host = $c->local_host();
```

obj: $c ("Apache2::Connection object")

ret: $local_host (string)

since: 2.0.00

META: you probably shouldn't use this method, but ("get_server_name") if inside request and $r is available.

"local_ip"

server IP address

```
$local_ip = $c->local_ip();
```

obj: $c ("Apache2::Connection object")

ret: $local_ip (string)

since: 2.0.00

"notes"

Get/set text notes for the duration of this connection. These notes can be passed from one module to another (not only mod_perl, but modules in any other language):

```
$notes      = $c->notes();
$prev_notes = $c->notes($new_notes);
```

obj: $c ("Apache2::Connection object")

opt arg1: $new_notes ("APR::Table object")

ret: $notes ("APR::Table object")

the current notes table.

if the $new_notes argument was passed, returns the previous value.

since: 2.0.00

Also see "$r->notes"

"output_filters"

Get the first filter in a linked list of protocol level output filters:

```
$output_filters = $c->output_filters();
$prev_output_filters = $r->output_filters($new_output_filters);
```

obj: $c ("Apache2::Connection object")

opt arg1: $new_output_filters

Set a new value

ret: $output_filters ("Apache2::Filter object")

The first filter in the connection output filters chain.

If $new_output_filters was passed, returns the previous value.

since: 2.0.00

For an example see: Bucket Brigades-based Protocol Module

"pool"

Pool associated with this connection

```
$p = $c->pool();
```

obj: $c ("Apache2::Connection object")

ret: $p ("APR::Pool object")

since: 2.0.00

"remote_addr"

Get this connection's remote socket address

```
$remote_sa = $c->remote_addr();
```

obj: $c ("Apache2::Connection object")

ret: $remote_sa ("APR::SockAddr object")

since: 2.0.00

"remote_ip"

Client's IP address

```
$remote_ip      = $c->remote_ip();
$prev_remote_ip = $c->remote_ip($new_remote_ip);
```

obj: $c ("Apache2::Connection object")

opt arg1: $new_remote_ip (string)

If passed a new value will be set

ret: $remote_ip (string)

current remote ip address

if the optional argument $new_remote_ip was passed the previous value is returned.

since: 2.0.00

"remote_host"

Client's DNS name:

$remote_host = $c->remote_host();

obj: $c ("Apache2::Connection object")

ret: $remote_host (string/undef)

If "$c->get_remote_host" was run it returns the cached value, which is a client DNS name or "" if it wasn't found. If the check wasn't run -- "undef" is returned.

since: 2.0.00

It's best to to call "$c->get_remote_host" instead of directly accessing this variable.

Unsupported API

"Apache2::Connection" 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.

"conn_config"

Config vector containing pointers to connections per-server config structures

$ret = $c->conn_config();

obj: $c ("Apache2::Connection object")

ret: $ret ("Apache2::ConfVector object")

since: 2.0.00

"sbh"

META: Autogenerated - needs to be reviewed/completed

handle to scoreboard information for this connection

$sbh = $c->sbh();

obj: $c ("Apache2::Connection object")

ret: $sbh (XXX)

since: 2.0.00

META: Not sure how this can be used from mod_perl at the moment. Unless "Apache2::Scoreboard" is extended to provide a hook to read from this variable.

See Also

mod_perl 2.0 documentation.

Copyright

Authors

The mod_perl development team and numerous contributors.