1 Apache - Perl interface to the Apache server API

1.1 Synopsis

```
use Apache ();
```

1.2 Description

This module provides a Perl interface the Apache API. It is here mainly for **mod_perl**, but may be used for other Apache modules that wish to embed a Perl interpreter. We suggest that you also consult the description of the Apache C API at http://httpd.apache.org/docs/ .

1.3 The Request Object

The request object holds all the information that the server needs to service a request. Apache **Perl*Handler**s will be given a reference to the request object as parameter and may choose to update or use it in various ways. Most of the methods described below obtain information from or update the request object. The perl version of the request object will be blessed into the **Apache** package, it is really a `request_rec*` in disguise.

1.3.1 Apache->request([$r])

The `Apache->request` method will return a reference to the request object.

Perl*Handlers can obtain a reference to the request object when it is passed to them via `@_`. However, scripts that run under **Apache::Registry**, for example, need a way to access the request object. **Apache::Registry** will make a request object available to these scripts by passing an object reference to `Apache->request($r)`. If handlers use modules such as **CGI::Apache** that need to access `Apache->request`, they too should do this (e.g. **Apache::Status**).

1.3.2 $r->as_string

Returns a string representation of the request object. Mainly useful for debugging.

1.3.3 $r->main

If the current request is a sub-request, this method returns a blessed reference to the main request structure. If the current request is the main request, then this method returns `undef`.

1.3.4 $r->prev

This method returns a blessed reference to the previous (internal) request structure or `undef` if there is no previous request.

1.3.5 $r->next

This method returns a blessed reference to the next (internal) request structure or `undef` if there is no next request.

1.3.6 $r->last

This method returns a blessed reference to the last (internal) request structure. Handy for logging modules.

1.3.7 $r->is_main

Returns true if the current request object is for the main request. (Should give the same result as `!$r->main`, but will be more efficient.)

1.3.8 $r->is_initial_req

Returns true if the current request is the first internal request, returns false if the request is a sub-request or internal redirect.

1.3.9 $r->allowed($bitmask)

Get or set the allowed methods bitmask. This allowed bitmask should be set whenever a 405 (method not allowed) or 501 (method not implemented) answer is returned. The bit corresponding to the method number should be set.

```
unless ($r->method_number == M_GET) {
  $r->allowed($r->allowed | (1<<M_GET) | (1<<M_HEAD) | (1<<M_OPTIONS));
  return HTTP_METHOD_NOT_ALLOWED;
}
```

1.4 Sub Requests

Apache provides a sub-request mechanism to lookup a uri or filename, performing all access checks, etc., without actually running the response phase of the given request. Notice, we have dropped the `sub_req_` prefix here. The `request_rec*` returned by the lookup methods is blessed into the **Apache::SubRequest** class. This way, `destroy_sub_request()` is called automatically during `Apache::SubRequest->DESTROY` when the object goes out of scope. The **Apache::SubRequest** class inherits all the methods from the **Apache** class.

1.4.1 $r->lookup_uri($uri)

```
my $subr = $r->lookup_uri($uri);
my $filename = $subr->filename;

unless(-e $filename) {
    warn "can't stat $filename!\n";
}
```

1.4.2 $r->lookup_file($filename)

```
my $subr = $r->lookup_file($filename);
```

1.4.3 $subr->run

```
if($subr->run != OK) {
    $subr->log_error("something went wrong!");
}
```

1.5 Client Request Parameters

In this section we will take a look at various methods that can be used to retrieve the request parameters sent from the client. In the following examples, `$r` is a request object blessed into the **Apache** class, obtained by the first parameter passed to a handler subroutine or *Apache->request*

1.5.1 $r->method([$meth])

The `$r->method` method will return the request method. It will be a string such as `"GET"`, `"HEAD"` or `"POST"`. Passing an argument will set the method, mainly used for internal redirects.

1.5.2 $r->method_number([$num])

The `$r->method_number` method will return the request method number. The method numbers are defined by the `M_GET`, `M_POST`,... constants available from the **Apache::Constants** module. Passing an argument will set the `method_number`, mainly used for internal redirects and testing authorization restriction masks.

1.5.3 $r->bytes_sent

The number of bytes sent to the client, handy for logging, etc.

1.5.4 $r->the_request

The request line sent by the client, handy for logging, etc.

1.5.5 $r->proxyreq

Returns true if the request is proxy http. Mainly used during the filename translation stage of the request, which may be handled by a `PerlTransHandler`.

1.5.6 $r->header_only

Returns true if the client is asking for headers only, e.g. if the request method was **HEAD**.

1.5.7 $r->protocol

The `$r->protocol` method will return a string identifying the protocol that the client speaks. Typical values will be `"HTTP/1.0"` or `"HTTP/1.1"`.

1.5.8 $r->hostname

Returns the server host name, as set by full URI or `Host:` header.

1.5.9 $r->request_time

Returns the time that the request was made. The time is the local unix time in seconds since the epoch.

1.5.10 $r->uri([$uri])

The `$r->uri` method will return the requested URI minus optional query string, optionally changing it with the first argument.

1.5.11 $r->filename([$filename])

The `$r->filename` method will return the result of the *URI --> filename* translation, optionally changing it with the first argument if you happen to be doing the translation.

1.5.12 $r->location

The `$r->location` method will return the path of the <Location> section from which the current `Perl*Handler` is being called.

1.5.13 $r->path_info([$path_info])

The `$r->path_info` method will return what is left in the path after the *URI --> filename* translation, optionally changing it with the first argument if you happen to be doing the translation.

1.5.14 $r->args([$query_string])

The `$r->args` method will return the contents of the URI *query string*. When called in a scalar context, the entire string is returned. When called in a list context, a list of parsed *key => value* pairs are returned, i.e. it can be used like this:

```
$query = $r->args;
%in    = $r->args;
```

`$r->args` can also be used to set the *query string*. This can be useful when redirecting a POST request.

1.5.15 $r->headers_in

The `$r->headers_in` method will return a `%hash` of client request headers. This can be used to initialize a perl hash, or one could use the `$r->header_in()` method (described below) to retrieve a specific header value directly.

Will return a *HASH* reference blessed into the *Apache::Table* class when called in a scalar context with no "key" argument. This requires *Apache::Table*.

1.5.16 $r->header_in($header_name, [$value])

Return the value of a client header. Can be used like this:

```
$ct = $r->header_in("Content-type");
$r->header_in($key, $val); #set the value of header '$key'
```

1.5.17 $r->content

The `$r->content` method will return the entity body read from the client, but only if the request content type is `application/x-www-form-urlencoded`. When called in a scalar context, the entire string is returned. When called in a list context, a list of parsed *key => value* pairs are returned. **NOTE**: you can only ask for this once, as the entire body is read from the client.

1.5.18 $r->read($buf, $bytes_to_read, [$offset])

This method is used to read data from the client, looping until it gets all of `$bytes_to_read` or a timeout happens.

An offset may be specified to place the read data at some other place than the beginning of the string.

In addition, this method sets a timeout before reading with `$r->soft_timeout`.

1.5.19 $r->get_remote_host

Lookup the client's DNS 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 host-name is not known.

1.5.20 $r->get_remote_logname

Lookup the remote user's system name. Might return *undef* if the remote system is not running an RFC 1413 server or if the configuration directive **IdentityCheck** is not turned on.

1.5.21 $r->user([$user])

If an authentication check was successful, the authentication handler caches the user name here. Sets the user name to the optional first argument.

1.5.22 Apache::Connection

More information about the client can be obtained from the **Apache::Connection** object, as described below.

1.5.23 $c = $r->connection

The `$r->connection` method will return a reference to the request connection object (blessed into the **Apache::Connection** package). This is really a `conn_rec*` in disguise. The following methods can be used on the connection object:

1.5.23.1 $c->remote_host

If the configuration directive **HostNameLookups** is set to on: then the first time `$r->get_remote_host` is called the server does a DNS lookup to get the remote client's host name. The result is cached in `$c->remote_host` then returned. If the server was unable to resolve the remote client's host name this will be set to "". Subsequent calls to `$r->get_remote_host` return this cached value.

If the configuration directive **HostNameLookups** is set to off: calls to `$r->get_remote_host` return a string that contains the dotted decimal representation of the remote client's IP address. However this string is not cached, and `$c->remote_host` is undefined. So, it's best to to call `$r->get_remote_host` instead of directly accessing this variable.

1.5.23.2 $c->remote_ip

The dotted decimal representation of the remote client's IP address. This is set by the server when the connection record is created so is always defined.

You can also set this value by providing an argument to it. This is helpful if your server is behind a squid accelerator proxy which adds a *X-Forwarded-For* header.

1.5.23.3 $c->local_addr

A packed `SOCKADDR_IN` in the same format as returned by `Socket::pack_sockaddr_in`, containing the port and address on the local host that the remote client is connected to. This is set by the server when the connection record is created so it is always defined.

1.5.23.4 $c->remote_addr

A packed `SOCKADDR_IN` in the same format as returned by `Socket::pack_sockaddr_in`, containing the port and address on the remote host that the server is connected to. This is set by the server when the connection record is created so it is always defined.

Among other things, this can be used, together with `$c->local_addr`, to perform RFC1413 ident lookups on the remote client even when the configuration directive **IdentityCheck** is turned off.

Can be used like:

```
use Net::Ident qw (lookupFromInAddr);
...
my $remoteuser = lookupFromInAddr ($c->local_addr,
                                   $c->remote_addr, 2);
```

Note that the lookupFromInAddr interface does not currently exist in the **Net::Ident** module, but the author is planning on adding it soon.

1.5.23.5 $c->remote_logname

If the configuration directive **IdentityCheck** is set to on: then the first time `$r->get_remote_logname` is called the server does an RFC 1413 (ident) lookup to get the remote users system name. Generally for UNI* systems this is their login. The result is cached in `$c->remote_logname` then returned. Subsequent calls to `$r->get_remote_host` return the cached value.

If the configuration directive **IdentityCheck** is set to off: then `$r->get_remote_logname` does nothing and `$c->remote_logname` is always undefined.

1.5.23.6 $c->user([$user])

Deprecated, use `$r->user` instead.

1.5.23.7 $c->auth_type

Returns the authentication scheme that successfully authenticate `$c->user`, if any.

1.5.23.8 $c->aborted

Returns true if the client stopped talking to us.

1.5.23.9 $c->fileno([$direction])

Returns the client file descriptor. If $direction is 0, the input fd is returned. If $direction is not null or ommitted, the output fd is returned.

This can be used to detect client disconnect without doing any I/O, e.g. using `IO::Select`.

1.6 Server Configuration Information

The following methods are used to obtain information from server configuration and access control files.

1.6.1 $r->dir_config($key)

Returns the value of a per-directory variable specified by the `PerlSetVar` directive.

```
# <Location /foo/bar>
# PerlSetVar  Key  Value
# </Location>

my $val = $r->dir_config('Key');
```

Keys are case-insensitive.

Will return a *HASH* reference blessed into the *Apache::Table* class when called in a scalar context with no "key" argument. See *Apache::Table*.

1.6.2 $r->dir_config->get($key)

Returns the value of a per-directory array variable specified by the `PerlAddVar` directive.

```
# <Location /foo/bar>
# PerlAddVar  Key  Value1
# PerlAddVar  Key  Value2
# </Location>

my @val = $r->dir_config->get('Key');
```

Alternatively in your code you can extend the setting with:

```
$r->dir_config->add(Key => 'Value3');
```

Keys are case-insensitive.

Will return a *HASH* reference blessed into the *Apache::Table* class when called in a scalar context with no "key" argument. See *Apache::Table*.

1.6.3 $r->requires

Returns an array reference of hash references, containing information related to the **require** directive. This is normally used for access control, see `Apache::AuthzAge` for an example.

1.6.4 $r->auth_type

Returns a reference to the current value of the per directory configuration directive **AuthType**. Normally this would be set to `Basic` to use the basic authentication scheme defined in RFC 1945, *Hypertext Transfer Protocol -- HTTP/1.0*. However, you could set to something else and implement your own authentication scheme.

1.6.5 $r->auth_name

Returns a reference to the current value of the per directory configuration directive **AuthName**. The AuthName directive creates protection realm within the server document space. To quote RFC 1945 "These realms allow the protected resources on a server to be partitioned into a set of protection spaces, each with its own authentication scheme and/or authorization database." The client uses the root URL of the server to determine which authentication credentials to send with each HTTP request. These credentials are tagged with the name of the authentication realm that created them. Then during the authentication stage the server uses the current authentication realm, from `$r->auth_name`, to determine which set of credentials to authenticate.

1.6.6 $r->document_root ([$docroot])

When called with no argument, returns a reference to the current value of the per server configuration directive **DocumentRoot**. To quote the Apache server documentation, "Unless matched by a directive like Alias, the server appends the path from the requested URL to the document root to make the path to the document." This same value is passed to CGI scripts in the `DOCUMENT_ROOT` environment variable.

You can also set this value by providing an argument to it. The following example dynamically sets the document root based on the request's "Host:" header:

```
sub trans_handler
  {
     my $r = shift;
     my ($user) = ($r->header_in('Host') =~ /^[^\.]+/);
     $r->document_root("/home/$user/www");
     return DECLINED;
  }

PerlTransHandler trans_handler
```

1.6.7 $r->server_root_relative([$relative_path])

If called without any arguments, this method returns the value of the currently-configured `ServerRoot` directory.

If a single argument is passed, it concatenates it with the value of `ServerRoot`. For example here is how to get the path to the *error_log* file under the server root:

```
my $error_log = $r->server_root_relative("logs/error_log");
```

See also the next item.

1.6.8 Apache->server_root_relative([$relative_path])

Same as the previous item, but this time it's used without a request object. This method is usually needed in a startup file. For example the following startup file modifies `@INC` to add a local directory with perl modules located under the server root and after that loads a module from that directory.

```
BEGIN {
    use Apache():
    use lib Apache->server_root_relative("lib/my_project");
}
use MyProject::Config ();
```

1.6.9 $r->allow_options

The `$r->allow_options` method can be used for checking if it is OK to run a perl script. The **Apache::Options** module provides the constants to check against.

```
if(!($r->allow_options & OPT_EXECCGI)) {
    $r->log_reason("Options ExecCGI is off in this directory",
                   $filename);
}
```

1.6.10 $r->get_server_port

Returns the port number on which the server is listening.

1.6.11 $s = $r->server

Return a reference to the server info object (blessed into the **Apache::Server** package). This is really a `server_rec*` in disguise. The following methods can be used on the server object:

1.6.12 $s = Apache->server

Same as above, but only available during server startup for use in `<Perl>` sections, **PerlRequire** or **PerlModule**.

1.6.13 $s->server_admin

Returns the mail address of the person responsible for this server.

1.6.14 $s->server_hostname

Returns the hostname used by this server.

1.6.15 $s->port

Returns the port that this servers listens too.

1.6.16 $s->is_virtual

Returns true if this is a virtual server.

1.6.17 $s->names

Returns the wild-carded names for ServerAlias servers.

1.6.18 $s->dir_config($key)

Alias for `Apache::dir_config`.

1.6.19 $s->warn

Alias for `Apache::warn`.

1.6.20 $s->log_error

Alias for `Apache::log_error`.

1.6.21 $s->uid

Returns the numeric user id under which the server answers requests. This is the value of the User directive.

1.6.22 $s->gid

Returns the numeric group id under which the server answers requests. This is the value of the Group directive.

1.6.23 $s->loglevel

Get or set the value of the current LogLevel. This method is added by the `Apache::Log` module, which needs to be pulled in.

```
use Apache::Log;
print "LogLevel = ", $s->loglevel;
$s->loglevel(Apache::Log::DEBUG);
```

If using Perl 5.005+, the following constants are defined (but not exported):

```
Apache::Log::EMERG
Apache::Log::ALERT
Apache::Log::CRIT
Apache::Log::ERR
Apache::Log::WARNING
Apache::Log::NOTICE
Apache::Log::INFO
Apache::Log::DEBUG
```

1.6.24 $r->get_handlers($hook)

Returns a reference to a list of handlers enabled for $hook. $hook is a string representing the phase to handle. The returned list is a list of references to the handler subroutines.

```
$list = $r->get_handlers( 'PerlHandler' );
```

1.6.25 $r->set_handlers($hook, [\&handler, ...])

Sets the list of handlers to be called for $hook. $hook is a string representing the phase to handle. The list of handlers is an anonymous array of code references to the handlers to install for this request phase. The special list `[ \&OK ]` can be used to disable a particular phase.

```
$r->set_handlers( PerlLogHandler => [ \&myhandler1, \&myhandler2 ] );
$r->set_handlers( PerlAuthenHandler => [ \&OK ] );
```

1.6.26 $r->push_handlers($hook, \&handler)

Pushes a new handler to be called for `$hook`. `$hook` is a string representing the phase to handle. The handler is a reference to a subroutine to install for this request phase. This handler will be called before any configured handlers.

```
$r->push_handlers( PerlHandler => \&footer);
```

1.6.27 $r->current_callback

Returns the name of the handler currently being run. This method is most useful to PerlDispatchHandlers who wish to only take action for certain phases.

```
if($r->current_callback eq "PerlLogHandler") {
        $r->warn("Logging request");
}
```

1.7 Setting Up the Response

The following methods are used to set up and return the response back to the client. This typically involves setting up `$r->status()`, the various content attributes and optionally some additional `$r->header_out()` calls before calling `$r->send_http_header()` which will actually send the headers to the client. After this a typical application will call the `$r->print()` method to send the response content to the client.

1.7.1 $r->send_http_header([$content_type])

Send the response line and all headers to the client. Takes an optional parameter indicating the content-type of the response, i.e. '`text/html`'.

This method will create headers from the `$r->content_xxx()` and `$r->no_cache()` attributes (described below) and then append the headers defined by $r->header_out (or `$r->err_header_out` if status indicates an error).

1.7.2 $r->get_basic_auth_pw

If the current request is protected by Basic authentication, this method will return OK. Otherwise, it will return a value that ought to be propagated back to the client (typically `AUTH_REQUIRED`). The second return value will be the decoded password sent by the client.

```
($ret, $sent_pw) = $r->get_basic_auth_pw;
```

1.7.3 $r->note_basic_auth_failure

Prior to requiring Basic authentication from the client, this method will set the outgoing HTTP headers asking the client to authenticate for the realm defined by the configuration directive `AuthName`.

1.7.4 $r->handler([$meth])

Set the handler for a request. Normally set by the configuration directive `AddHandler`.

```
$r->handler( "perl-script" );
```

1.7.5 $r->notes($key, [$value])

Return the value of a named entry in the Apache `notes` table, or optionally set the value of a named entry. This table is used by Apache modules to pass messages amongst themselves. Generally if you are writing handlers in mod_perl you can use Perl variables for this.

```
$r->notes("MY_HANDLER" => OK);
$val = $r->notes("MY_HANDLER");
```

Will return a *HASH* reference blessed into the *Apache::Table* class when called in a scalar context with no "key" argument. This requires *Apache::Table*.

1.7.6 $r->pnotes($key, [$value])

Like `$r->notes`, but takes any scalar as an value.

```
$r->pnotes("MY_HANDLER" => [qw(one two)]);
my $val = $r->pnotes("MY_HANDLER");
print $val->[0];     # prints "one"
```

Advantage over just using a Perl variable is that `$r->pnotes` gets cleaned up after every request.

1.7.7 $r->subprocess_env($key, [$value])

Return the value of a named entry in the Apache `subprocess_env` table, or optionally set the value of a named entry. This table is used by mod_include. By setting some custom variables inside a perl handler it is possible to combine perl with mod_include nicely. If you say, e.g. in a PerlHeaderParserHandler

```
$r->subprocess_env(MyLanguage => "de");
```

you can then write in your .shtml document:

```
<!--#if expr="$MyLanguage = en" -->
English
<!--#elif expr="$MyLanguage = de" -->
Deutsch
<!--#else -->
Sorry
<!--#endif -->
```

Will return a *HASH* reference blessed into the *Apache::Table* class when called in a scalar context with no "key" argument. This requires *Apache::Table*.

1.7.8 $r->content_type([$newval])

Get or set the content type being sent to the client. Content types are strings like `"text/plain"`, `"text/html"` or `"image/gif"`. This corresponds to the `"Content-Type"` header in the HTTP protocol. Example of usage is:

```
$previous_type = $r->content_type;
$r->content_type("text/plain");
```

1.7.9 $r->content_encoding([$newval])

Get or set the content encoding. Content encodings are string like "gzip" or "compress". This correspond to the "Content-Encoding" header in the HTTP protocol.

1.7.10 $r->content_languages([$array_ref])

Get or set the content languages. The content language corresponds to the "Content-Language" HTTP header and is an array reference containing strings such as "en" or "no".

1.7.11 $r->status($integer)

Get or set the reply status for the client request. The **Apache::Constants** module provide mnemonic names for the status codes.

1.7.12 $r->status_line($string)

Get or set the response status line. The status line is a string like "200 Document follows" and it will take precedence over the value specified using the `$r->status()` described above.

1.7.13 $r->headers_out

The `$r->headers_out` method will return a `%hash` of server response headers. This can be used to initialize a perl hash, or one could use the `$r->header_out()` method (described below) to retrieve or set a specific header value directly.

Will return a *HASH* reference blessed into the *Apache::Table* class when called in a scalar context with no "key" argument. This requires *Apache::Table*.

1.7.14 $r->header_out($header, $value)

Change the value of a response header, or create a new one. You should not define any "Content-XXX" headers by calling this method, because these headers use their own specific methods. Example of use:

```
$r->header_out("WWW-Authenticate" => "Basic");
$val = $r->header_out($key);
```

1.7.15 $r->err_headers_out

The `$r->err_headers_out` method will return a %hash of server response headers. This can be used to initialize a perl hash, or one could use the `$r->err_header_out()` method (described below) to retrieve or set a specific header value directly.

The difference between headers_out and err_headers_out is that the latter are printed even on error, and persist across internal redirects (so the headers printed for ErrorDocument handlers will have them).

Will return a *HASH* reference blessed into the *Apache::Table* class when called in a scalar context with no "key" argument. This requires *Apache::Table*.

1.7.16 $r->err_header_out($header, [$value])

Change the value of an error response header, or create a new one. These headers are used if the status indicates an error.

```
$r->err_header_out("Warning" => "Bad luck");
$val = $r->err_header_out($key);
```

1.7.17 $r->no_cache($boolean)

This is a flag that indicates that the data being returned is volatile and the client should be told not to cache it. `$r->no_cache(1)` adds the headers `"Pragma: no-cache"` and `"Cache-control: no-cache"` to the reponse, therefore it must be called before `$r->send_http_header`.

1.7.18 $r->print(@list)

This method sends data to the client with `$r->write_client`, but first sets a timeout before sending with `$r->soft_timeout`. This method is called instead of `CORE::print` when you use `print()` in your mod_perl programs.

This method treats scalar references specially. If an item in `@list` is a scalar reference, it will be dereferenced before printing. This is a performance optimization which prevents unneeded copying of large strings, and it is subtly different from Perl's standard `print()` behavior.

Example:

```
$foo = \"bar"; print($foo);
```

The result is "bar", not the "SCALAR(0xDEADBEEF)" you might have expected. If you really want the reference to be printed out, force it into a scalar context by using `print(scalar($foo))`.

The print-a-scalar-reference feature is now deprecated. There are known bugs when using it and it's not supported by mod_perl 2.0. If you have a scalar reference containing a string to be printed, dereference it before sending it to print.

1.7.19 $r->send_fd($filehandle)

Send the contents of a file to the client. Can for instance be used like this:

```
open(FILE, $r->filename) || return 404;
$r->send_fd(FILE);
close(FILE);
```

1.7.20 $r->internal_redirect($newplace)

Redirect to a location in the server namespace without telling the client. For instance:

```
$r->internal_redirect("/home/sweet/home.html");
```

1.7.21 $r->internal_redirect_handler($newplace)

Same as *internal_redirect*, but the *handler* from `$r` is preserved.

1.7.22 $r->custom_response($code, $uri)

This method provides a hook into the **ErrorDocument** mechanism, allowing you to configure a custom response for a given response code at request-time.

Example:

```
use Apache::Constants ':common';

sub handler {
    my ($r) = @_;

    if($things_are_ok) {
        return OK;
    }

    #<Location $r->uri>
    #ErrorDocument 401 /error.html
    #</Location>

    $r->custom_response(AUTH_REQUIRED, "/error.html");

    #can send a string too
    #<Location $r->uri>
    #ErrorDocument 401 "sorry, go away"
    #</Location>
```

```
        #$r->custom_response(AUTH_REQUIRED, "sorry, go away");

        return AUTH_REQUIRED;
    }
```

1.8 Server Core Functions

1.8.1 $r->soft_timeout($message)

1.8.2 $r->hard_timeout($message)

1.8.3 $r->kill_timeout

1.8.4 $r->reset_timeout

(Documentation borrowed from http_main.h)

There are two functions which modules can call to trigger a timeout (with the per-virtual-server timeout duration); these are `hard_timeout` and `soft_timeout`.

The difference between the two is what happens when the timeout expires (or earlier than that, if the client connection aborts) --- a `soft_timeout` just puts the connection to the client in an "aborted" state, which will cause *http_protocol.c* to stop trying to talk to the client, but otherwise allows the code to continue normally. `hard_timeout()`, by contrast, logs the request, and then aborts it completely --- `longjmp()`ing out to the `accept()` loop in `http_main`. Any resources tied into the request resource pool will be cleaned up; everything that is not will leak.

`soft_timeout()` is recommended as a general rule, because it gives your code a chance to clean up. However, `hard_timeout()` may be the most convenient way of dealing with timeouts waiting for some external resource other than the client, if you can live with the restrictions.

When a hard timeout is in scope, critical sections can be guarded with `block_alarms()` and `unblock_alarms()` --- these are declared in *alloc.c* because they are most often used in conjunction with routines to allocate something or other, to make sure that the cleanup does get registered before any alarm is allowed to happen which might require it to be cleaned up; they * are, however, implemented in *http_main.c*.

`kill_timeout()` will disarm either variety of timeout.

`reset_timeout()` resets the timeout in progress.

1.8.5 $r->post_connection($code_ref)

1.8.6 $r->register_cleanup($code_ref)

Register a cleanup function which is called just before $r->pool is destroyed.

```
$r->register_cleanup(sub {
    my $r = shift;
    warn "registered cleanup called for ", $r->uri, "\n";
});
```

Cleanup functions registered in the parent process (before forking) will run once when the server is shut down:

```
#PerlRequire startup.pl
warn "parent pid is $$\n";
Apache->server->register_cleanup(sub { warn "server cleanup in $$\n"});
```

The *post_connection* method is simply an alias for *register_cleanup*, as this method may be used to run code after the client connection is closed, which may not be a *cleanup*.

1.9 CGI Support

We also provide some methods that make it easier to support the CGI type of interface.

1.9.1 $r->send_cgi_header()

Take action on certain headers including *Status:*, *Location:* and *Content-type:* just as mod_cgi does, then calls $r->send_http_header(). Example of use:

```
$r->send_cgi_header(<<EOT);
Location: /foo/bar
Content-type: text/html

EOT
```

1.10 Error Logging

The following methods can be used to log errors.

1.10.1 $r->log_reason($message, $file)

The request failed, why?? Write a message to the server errorlog.

```
$r->log_reason("Because I felt like it", $r->filename);
```

1.10.2 $r->log_error($message)

Uh, oh. Write a message to the server errorlog.

```
$r->log_error("Some text that goes in the error_log");
```

1.10.3 $r->warn($message)

For pre-1.3 versions of apache, this is just an alias for `log_error`. With 1.3+ versions of apache, this message will only be send to the error_log if **LogLevel** is set to **warn** or higher.

1.11 Utility Functions

1.11.1 Apache::unescape_url($string)

```
$unescaped_url = Apache::unescape_url($string)
```

Handy function for unescapes. Use this one for filenames/paths. Notice that the original `$string` is mangled in the process (because the string part of PV shrinks, but the variable is not updated, to speed things up).

Use `unescape_url_info` for the result of submitted form data.

1.11.2 Apache::unescape_url_info($string)

Handy function for unescapes submitted form data. In opposite to `unescape_url` it translates the plus sign to space.

1.11.3 Apache::perl_hook($hook)

Returns true if the specified callback hook is enabled:

```
for (qw(Access Authen Authz ChildInit Cleanup Fixup
        HeaderParser Init Log Trans Type))
{
    print "$_ hook enabled\n" if Apache::perl_hook($_);
}
```

1.12 Global Variables

1.12.1 $Apache::Server::Starting

Set to true when the server is starting.

1.12.2 $Apache::Server::ReStarting

Set to true when the server is starting.

1.13 See Also

perl, Apache::Constants, Apache::Registry, `Apache::Debug`, Apache::Options, CGI

Apache C API notes at http://httpd.apache.org/docs/

1.14 Maintainers

Maintainer is the person(s) you should contact with updates, corrections and patches.

- Adam Pisoni <adam@cnation.com>, but contact modperl docs list

1.15 Authors

Perl interface to the Apache C API written by Doug MacEachern with contributions from Gisle Aas, Andreas Koenig, Eric Bartley, Rob Hartill, Gerald Richter, Salvador Ortiz and others.

Table of Contents: