mod_perl 1.0 API

Here is the documentation for the whole API provided with the mod_perl distribution, ie. of various Apache:: modules you will need to use.

Last modified Sun Feb 16 01:33:53 2014 GMT

Part I: Access to the Apache API

- 1. Apache - Perl interface to the Apache server API
  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/ .

- 2. Apache::Constants - Constants defined in apache header files
  Server constants used by apache modules are defined in **httpd.h** and other header files, this module gives Perl access to those constants.

- 3. Apache::Options - OPT_* defines from httpd_core.h
  The `Apache::Options` module will export the following bitmask constants:

- 4. Apache::Table - Perl interface to the Apache table structure
  This module provides tied interfaces to Apache data structures.

- 5. Apache::File - advanced functions for manipulating files at the server side
  `Apache::File` does two things: it provides an object-oriented interface to filehandles similar to Perl's standard `IO::File class`. While the `Apache::File` module does not provide all the functionality of `IO::File`, its methods are approximately twice as fast as the equivalent `IO::File` methods. Secondly, when you use `Apache::File`, it adds several new methods to the `Apache` class which provide support for handling files under the `HTTP/1.1` protocol.

- 6. Apache::Log - Interface to Apache logging
  The `Apache::Log` module provides an interface to Apache's *ap_log_error* and *ap_log_rerror* routines.

- 7. Apache::URI - URI component parsing and unparsing
  This module provides an interface to the Apache *util_uri* module and the *uri_components* structure.

- 8. Apache::Util - Interface to Apache C util functions
  This module provides a Perl interface to some of the C utility functions available in Apache. The same functionality is avaliable in libwww-perl, but the C versions are faster:

- 9. Apache::Include - Utilities for mod_perl/mod_include integration
  The `Apache::Include` module provides a handler, making it simple to include `Apache::Registry` scripts with the mod_include perl directive.

Part II: Run CGI scripts under mod_perl

- 10. Apache::Registry - Run unaltered CGI scrips under mod_perl
  `Apache::Registry` is the Apache module allowing you to run CGI scripts very fast under mod_perl, by compiling all scripts once and then caching them in memory.

- 11. Apache::PerlRun - Run unaltered CGI scripts under mod_perl
  This module's `handler` emulates the CGI environment, allowing programmers to write scripts that run under CGI or mod_perl without change. Unlike `Apache::Registry`, the `Apache::PerlRun` handler does not cache the script inside of a subroutine. Scripts will be

"compiled" every request. After the script has run, it's namespace is flushed of all variables and subroutines.

- 12. Apache::RegistryLoader - Compile Apache::Registry scripts at server startup
This modules allows compilation of `Apache::Registry` scripts at server startup.

Part III: Development/Debugging help

- 13. Apache::StatINC - Reload %INC files when updated on disk
When Perl pulls a file via `require`, it stores the filename in the global hash `%INC`. The next time Perl tries to `require` the same file, it sees the file in `%INC` and does not reload from disk. This module's handler iterates over `%INC` and reloads the file if it has changed on disk.

- 14. Apache::test - Facilitates testing of Apache::* modules
This module helps authors of `Apache::*` modules write test suites that can query an actual running Apache server with mod_perl and their modules loaded into it.

- 15. Apache::Symdump - Symbol table snapshots
`Apache::Symdump` will record snapshots of the Perl symbol table for you to look at later.

- 16. Apache::src - Methods for locating and parsing bits of Apache source code
This module provides methods for locating and parsing bits of Apache source code.

- 17. Apache::Leak - Module for tracking memory leaks in mod_perl code
`Apache::Leak` is a module built to track memory leaks in mod_perl code.

- 18. Apache::FakeRequest - fake request object for debugging
`Apache::FakeRequest` is used to set up an empty Apache request object that can be used for debugging.

- 19. Apache::Debug - Utilities for debugging embedded perl code
This module sends what may be helpful debugging info to the client rather that the error log.

- 20. Apache::Symbol - Things for symbol things
`Apache::Symbol` helps mod_perl users avoid Perl warnings related with redefined constant functions.

- 21. Apache::SIG - Override apache signal handlers with Perl's
When a client drops a connection and apache is in the middle of a write, a timeout will occur and httpd sends a `SIGPIPE`. When apache's `SIGPIPE` handler is used, Perl may be left in the middle of it's eval context, causing bizarre errors during subsequent requests are handled by that child. When `Apache::SIG` is used, it installs a different `SIGPIPE` handler which rewinds the context to make sure Perl is back to normal state, preventing these bizarre errors.

Part IV: Apache configuration

- 22. Apache::PerlSections - Utilities for work with Perl sections
It is possible to configure you server entirely in Perl using `<Perl>` sections in *httpd.conf*. This module is here to help you with such a task.

- 23. Apache::httpd_conf - Generate an httpd.conf file
  The Apache::httpd_conf module will generate a tiny httpd.conf file, which pulls itself back in via a `<Perl>` section.

- 24. Apache::Status - Embedded interpreter status information
  The `Apache::Status` module provides some information about the status of the Perl interpreter embedded in the server.

Part V: Server Maintenance

- 25. Apache::Resource - Limit resources used by httpd children
  `Apache::Resource` uses the `BSD::Resource` module, which uses the C function `setrlimit` to set limits on system resources such as memory and cpu usage.

- 26. Apache::SizeLimit - Because size does matter.
  This module allows you to kill off Apache httpd processes if they grow too large.

See perldoc.perl.org for documentation of the rest of the `Apache::` modules

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 hostname 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.

2 Apache::Constants - Constants defined in apache header files

2.1 Synopsis

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

2.2 Description

Server constants used by apache modules are defined in **httpd.h** and other header files, this module gives Perl access to those constants.

2.3 Export Tags

- **common**

  This tag imports the most commonly used constants.

  ```
  OK
  DECLINED
  DONE
  NOT_FOUND
  FORBIDDEN
  AUTH_REQUIRED
  SERVER_ERROR
  ```

- **response**

  This tag imports the **common** response codes, plus these response codes:

  ```
  DOCUMENT_FOLLOWS
  MOVED
  REDIRECT
  USE_LOCAL_COPY
  BAD_REQUEST
  BAD_GATEWAY
  RESPONSE_CODES
  NOT_IMPLEMENTED
  CONTINUE
  NOT_AUTHORITATIVE
  ```

  CONTINUE and **NOT_AUTHORITATIVE** are aliases for **DECLINED**.

- **methods**

  This are the method numbers, commonly used with the Apache **method_number** method.

  ```
  METHODS
  M_CONNECT
  M_DELETE
  M_GET
  M_INVALID
  ```

```
M_OPTIONS
M_POST
M_PUT
M_TRACE
M_PATCH
M_PROPFIND
M_PROPPATCH
M_MKCOL
M_COPY
M_MOVE
M_LOCK
M_UNLOCK
```

- **options**

  These constants are most commonly used with the Apache **allow_options** method:

  ```
  OPT_NONE
  OPT_INDEXES
  OPT_INCLUDES
  OPT_SYM_LINKS
  OPT_EXECCGI
  OPT_UNSET
  OPT_INCNOEXEC
  OPT_SYM_OWNER
  OPT_MULTI
  OPT_ALL
  ```

- **satisfy**

  These constants are most commonly used with the Apache **satisfies** method:

  ```
  SATISFY_ALL
  SATISFY_ANY
  SATISFY_NOSPEC
  ```

- **remotehost**

  These constants are most commonly used with the Apache **get_remote_host** method:

  ```
  REMOTE_HOST
  REMOTE_NAME
  REMOTE_NOLOOKUP
  REMOTE_DOUBLE_REV
  ```

- **http**

  This is the full set of HTTP response codes: (NOTE: not all implemented here)

  ```
  HTTP_OK
  HTTP_MOVED_TEMPORARILY
  HTTP_MOVED_PERMANENTLY
  HTTP_METHOD_NOT_ALLOWED
  HTTP_NOT_MODIFIED
  HTTP_UNAUTHORIZED
  HTTP_FORBIDDEN
  ```

```
HTTP_NOT_FOUND
HTTP_BAD_REQUEST
HTTP_INTERNAL_SERVER_ERROR
HTTP_NOT_ACCEPTABLE
HTTP_NO_CONTENT
HTTP_PRECONDITION_FAILED
HTTP_SERVICE_UNAVAILABLE
HTTP_VARIANT_ALSO_VARIES
```

- **server**

  These are constants related to server version:

  ```
  MODULE_MAGIC_NUMBER
  SERVER_VERSION
  SERVER_BUILT
  ```

- **config**

  These are constants related to configuration directives:

  ```
  DECLINE_CMD
  ```

- **types**

  These are constants related to internal request types:

  ```
  DIR_MAGIC_TYPE
  ```

- **override**

  These constants are used to control and test the context of configuration directives.

  ```
  OR_NONE
  OR_LIMIT
  OR_OPTIONS
  OR_FILEINFO
  OR_AUTHCFG
  OR_INDEXES
  OR_UNSET
  OR_ALL
  ACCESS_CONF
  RSRC_CONF
  ```

- **args_how**

```
RAW_ARGS
TAKE1
TAKE2
TAKE12
TAKE3
TAKE23
TAKE123
ITERATE
ITERATE2
FLAG
NO_ARGS
```

2.4 Warnings

You should be aware of the issues relating to using constant subroutines in Perl. For example, look at this example:

```
$r->custom_response(FORBIDDEN => "File size exceeds quota.");
```

This will not set a custom response for `FORBIDDEN`, but for the string `"FORBIDDEN"`, which clearly isn't what is expected. You'll get an error like this:

```
[Tue Apr 23 19:46:14 2002] null: Argument "FORBIDDEN" isn't numeric in subroutine entry at ...
```

Therefore, you can avoid this by not using the hash notation for things that don't require it.

```
$r->custom_response(FORBIDDEN,  "File size exceeds quota.");
```

Another important note is that you should be using the correct constants defined here, and not direct HTTP codes. For example:

```
sub handler {
    return 200;
}
```

Is not correct. The correct use is:

```
use Apache::Constants qw(OK);

sub handler {
    return OK;
}
```

Also remember that `OK != HTTP_OK`.

2.5 Maintainers

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

- **The documentation mailing list**

2.6 Authors

- **Doug MacEachern**
- **Gisle Aas**
- **h2xs**

Only the major authors are listed above. For contributors see the Changes file.

3 Apache::Options - OPT_* defines from httpd_core.h

3.1 Synopsis

```
use Apache::Options;
```

3.2 Description

The `Apache::Options` module will export the following bitmask constants:

```
OPT_NONE
OPT_INDEXES
OPT_INCLUDES
OPT_SYMLINKS
OPT_EXECCGI
OPT_UNSET
OPT_INCNOEXEC
OPT_SYM_OWNER
OPT_MULTI
OPT_ALL
```

These constants can be used to check the return value from `Apache->request->allow_options()` method.

This module is simply a stub which imports from `Apache::Constants`, just as if you had said `use Apache::Constants ':options';`.

3.3 Maintainers

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

- **The documentation mailing list**

3.4 Authors

- **Doug MacEachern**

Only the major authors are listed above. For contributors see the Changes file.

3.5 See Also

Apache, Apache::Constants

4 Apache::Table - Perl interface to the Apache table structure

4.1 Synopsis

```
use Apache::Table ();

my $headers_out = $r->headers_out;
while(my ($key,$val) = each %$headers_out) {
  ...
}

my $table = $r->headers_out;
$table->set(From => 'dougm@perl.apache.org');
```

mod_perl needs to be compiled with at least one of the following options:

```
DYNAMIC=1
PERL_TABLE_API=1
EVERYTHING=1
```

4.2 Description

This module provides tied interfaces to Apache data structures.

4.2.1 Classes

- **Apache::Table**

  The `Apache::Table` class provides methods for interfacing with the Apache `table` structure. The following `Apache` class methods, when called in a scalar context with no "key" argument, will return a *HASH* reference blessed into the *Apache::Table* class and where *HASH* is tied to `Apache::Table`:

  ```
  headers_in
  headers_out
  err_headers_out
  notes
  dir_config
  subprocess_env
  ```

4.2.2 Methods

- **get**

  Corresponds to the `ap_table_get` function.

  ```
  my $value = $table->get($key);

  my $value = $headers_out->{$key};
  ```

- **set**

Corresponds to the `ap_table_set` function.

```
$table->set($key, $value);

$headers_out->{$key} = $value;
```

- **unset**

  Corresponds to the `ap_table_unset` function.

  ```
  $table->unset($key);

  delete $headers_out->{$key};
  ```

- **clear**

  Corresponds to the `ap_table_clear` function.

  ```
  $table->clear;

  %$headers_out = ();
  ```

- **add**

  Corresponds to the `ap_table_add` function.

  ```
  $table->add($key, $value);
  ```

- **merge**

  Corresponds to the `ap_table_merge` function.

  ```
  $table->merge($key, $value);
  ```

4.3 Maintainers

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

- **The documentation mailing list**

4.4 Authors

- **Doug MacEachern**

Only the major authors are listed above. For contributors see the Changes file.

4.5 See Also

Apache, mod_perl

5 Apache::File - advanced functions for manipulating files at the server side

5 Apache::File - advanced functions for manipulating files at the server side

5.1 Synopsis

```
use Apache::File ();

my $fh = Apache::File->new($filename);
print $fh 'Hello';
$fh->close;

my ($name, $fh) = Apache::File->tmpfile;

if ((my $rc = $r->discard_request_body) != OK) {
   return $rc;
}

if((my $rc = $r->meets_conditions) != OK) {
   return $rc;
}

my $date_string = localtime $r->mtime;

$r->set_content_length;
$r->set_etag;
$r->update_mtime;
$r->set_last_modified;
```

5.2 Description

`Apache::File` does two things: it provides an object-oriented interface to filehandles similar to Perl's standard `IO::File class`. While the `Apache::File` module does not provide all the functionality of `IO::File`, its methods are approximately twice as fast as the equivalent `IO::File` methods. Secondly, when you use `Apache::File`, it adds several new methods to the `Apache` class which provide support for handling files under the `HTTP/1.1` protocol.

5.3 Apache::File methods

- **new()**

  This method creates a new filehandle, returning the filehandle object on success, undef on failure. If an additional argument is given, it will be passed to the `open()` method automatically.

  ```
  use Apache::File ();
  my $fh = Apache::File->new;

  my $fh = Apache::File->new($filename) or die "Can't open $filename $!";
  ```

- **open()**

  Given an Apache::File object previously created with `new()`, this method opens a file and associates it with the object. The `open()` method accepts the same types of arguments as the standard Perl `open()` function, including support for file modes.

```
$fh->open($filename);

$fh->open(">$out_file");

$fh->open("|$program");
```

- **close()**

  The `close()` method is equivalent to the Perl builtin close function, returns true upon success, false upon failure.

  ```
  $fh->close or die "Can't close $filename $!";
  ```

- **tmpfile()**

  The `tmpfile()` method is responsible for opening up a unique temporary file. It is similar to the `tmpnam()` function in the `POSIX` module, but doesn't come with all the memory overhead that loading `POSIX` does. It will choose a suitable temporary directory (which must be writable by the Web server process). It then generates a series of filenames using the current process ID and the `$TMPNAM` package global. Once a unique name is found, it is opened for writing, using flags that will cause the file to be created only if it does not already exist. This prevents race conditions in which the function finds what seems to be an unused name, but someone else claims the same name before it can be created.

  As an added bonus, `tmpfile()` calls the `register_cleanup()` method behind the scenes to make sure the file is unlinked after the transaction is finished.

  Called in a list context, `tmpfile()` returns the temporary file name and a filehandle opened for reading and writing. In a scalar context only the filehandle is returned.

  ```
  my ($tmpnam, $fh) = Apache::File->tmpfile;

  my $fh = Apache::File->tmpfile;
  ```

5.4 Apache Methods added by Apache::File

When a handler pulls in `Apache::File`, the module adds a number of new methods to the Apache request object. These methods are generally of interest to handlers that wish to serve static files from disk or memory using the features of the `HTTP/1.1` protocol that provide increased performance through client-side document caching.

- **$r->discard_request_body()**

  This method tests for the existence of a request body and if present, simply throws away the data. This discarding is especially important when persistent connections are being used, so that the request body will not be attached to the next request. If the request is malformed, an error code will be returned, which the module handler should propagate back to Apache.

```
if ((my $rc = $r->discard_request_body) != OK) {
   return $rc;
}
```

- **$r->meets_conditions()**

  In the interest of HTTP/1.1 compliance, the `meets_conditions()` method is used to implement ‘‘conditional GET’’ rules. These rules include inspection of client headers, including `If-Modified-Since`, `If-Unmodified-Since`, `If-Match` and `If-None-Match`.

  As far as Apache modules are concerned, they need only check the return value of this method before sending a request body. If the return value is anything other than `OK`, the module should return from the handler with that value. A common return value other than `OK` is `HTTP_NOT_MODIFIED`, which is sent when the document is already cached on the client side, and has not changed since it was cached.

  ```
  if((my $rc = $r->meets_conditions) != OK) {
     return $rc;
  }
  #else ... go and send the response body ...
  ```

- **$r->mtime()**

  This method returns the last modified time of the requested file, expressed as seconds since the epoch.

  ```
  my $date_string = localtime $r->mtime;
  ```

  To change the last modified time use the `update_mtime()` method.

- **$r->set_content_length()**

  This method sets the outgoing `Content-length` header based on its argument, which should be expressed in byte units. If no argument is specified, the method will use the size returned by `$r->filename`. This method is a bit faster and more concise than setting `Content-length` in the headers_out table yourself.

  ```
  $r->set_content_length;
  $r->set_content_length(-s $r->finfo); #same as above
  $r->set_content_length(-s $filename);
  ```

- **$r->set_etag()**

  This method is used to set the outgoing `ETag` header corresponding to the requested file. `ETag` is an opaque string that identifies the currrent version of the file and changes whenever the file is modified. This string is tested by the `meets_conditions()` method if the client provide an `If-Match` or `If-None-Match` header.

  ```
  $r->set_etag;
  ```

- **$r->set_last_modified()**

This method is used to set the outgoing `Last-Modified header` from the value returned by `$r->mtime`. The method checks that the specified time is not in the future. In addition, using `set_last_modified()` is faster and more concise than setting `Last-Modified` in the `headers_out` table yourself.

You may provide an optional time argument, in which case the method will first call the `update_mtime()` to set the file's last modification date. It will then set the outgoing `Last-Modified` header as before.

```
$r->update_mtime((stat $r->finfo)[9]);
$r->set_last_modified;
$r->set_last_modified((stat $r->finfo)[9]); #same as the two lines above
```

- **$r->update_mtime()**

  Rather than setting the request record mtime field directly, you can use the `update_mtime()` method to change the value of this field. It will only be updated if the new time is more recent than the current mtime. If no time argument is present, the default is the last modified time of $r->filename.

```
$r->update_mtime;
$r->update_mtime((stat $r->finfo)[9]); #same as above
$r->update_mtime(time);
```

5.5 Maintainers

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

- **The documentation mailing list**

5.6 Authors

- **Doug MacEachern**

Only the major authors are listed above. For contributors see the Changes file.

6 Apache::Log - Interface to Apache logging

6.1 Synopsis

```
use Apache::Log ();
my $rlog = $r->log;
$rlog->debug("You only see this if 'LogLevel' is set to 'debug'");

my $slog = $r->server->log;
```

6.2 Description

The `Apache::Log` module provides an interface to Apache's *ap_log_error* and *ap_log_rerror* routines.

The methods listed below can be called as `$r->`*meth*`($error)`, and the error message will appear in the error log depending on the value of `LogLevel`.

- **emerg**
- **alert**
- **crit**
- **error**
- **warn**
- **notice**
- **info**
- **debug**

6.3 Maintainers

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

- **The documentation mailing list**

6.4 Authors

- **Doug MacEachern**

Only the major authors are listed above. For contributors see the Changes file.

6.5 See Also

mod_perl, Apache.

7 Apache::URI - URI component parsing and unparsing

7.1 Synopsis

```
use Apache::URI ();
my $uri = $r->parsed_uri;

my $uri = Apache::URI->parse($r, "http://perl.apache.org/");
```

7.2 Description

This module provides an interface to the Apache *util_uri* module and the *uri_components* structure.

7.3 Methods

- **Apache::parsed_uri**

  Apache will have already parsed the requested uri components, which can be obtained via the *parsed_uri* method defined in the *Apache* class. This method returns an object blessed into the *Apache::URI* class.

  ```
  my $uri = $r->parsed_uri;
  ```

- **parse**

  This method will parse a URI string into uri components which are stashed in the *Apache::URI* object it returns.

  ```
      my $uri = Apache::URI->parse($r, "http://www.foo.com/path/file.html?query+string");
  ```

  This method is considerably faster than using *URI::URL*:

  ```
      timethese(5000, {
  C => sub { Apache::URI->parse($r, $test_uri) },
  Perl => sub { URI::URL->new($test_uri) },
      });

   Benchmark: timing 5000 iterations of C, Perl...
     C:  1 secs ( 0.62 usr  0.04 sys =  0.66 cpu)
     Perl:  6 secs ( 6.21 usr  0.08 sys =  6.29 cpu)
  ```

- **unparse**

  This method will join the uri components back into a string version.

  ```
  my $string = $uri->unparse;
  ```

- **scheme**

  ```
  my $scheme = $uri->scheme;
  ```

- **hostinfo**

```
my $hostinfo = $uri->hostinfo;
```

- **user**

  ```
  my $user = $uri->user;
  ```

- **password**

  ```
  my $password = $uri->password;
  ```

- **hostname**

  ```
  my $hostname = $uri->hostname;
  ```

- **port**

  ```
  my $port = $uri->port;
  ```

- **path**

  ```
  my $path = $uri->path;
  ```

- **rpath**

  Returns the *path* minus *path_info*.

  ```
  my $path = $uri->rpath;
  ```

- **query**

  ```
  my $query = $uri->query;
  ```

- **fragment**

  ```
  my $fragment = $uri->fragment;
  ```

7.4 Author

Doug MacEachern

7.5 See Also

perl.

8 Apache::Util - Interface to Apache C util functions

8.1 Synopsis

```
use Apache::Util qw(:all);
```

8.2 Description

This module provides a Perl interface to some of the C utility functions available in Apache. The same functionality is avaliable in libwww-perl, but the C versions are faster:

```
use Benchmark;
timethese(1000, {
    C => sub { my $esc = Apache::Util::escape_html($html) },
    Perl => sub { my $esc = HTML::Entities::encode($html) },
});

Benchmark: timing 1000 iterations of C, Perl...
        C:  0 secs ( 0.17 usr  0.00 sys =  0.17 cpu)
     Perl: 15 secs (15.06 usr  0.04 sys = 15.10 cpu)

use Benchmark;
timethese(10000, {
    C => sub { my $esc = Apache::Util::escape_uri($uri) },
    Perl => sub { my $esc = URI::Escape::uri_escape($uri) },
});

Benchmark: timing 10000 iterations of C, Perl...
        C:  0 secs ( 0.55 usr  0.01 sys =  0.56 cpu)
     Perl:  2 secs ( 1.78 usr  0.01 sys =  1.79 cpu)
```

8.3 Functions

- **escape_html**

  This routine replaces unsafe characters in $string with their entity representation.

  ```
  my $esc = Apache::Util::escape_html($html);
  ```

  This function will correctly escape US-ASCII output. If you are using a different character set such as UTF8, or need more control on the escaping process, use HTML::Entities.

- **escape_uri**

  This function replaces all unsafe characters in the $string with their escape sequence and returns the result.

  ```
  my $esc = Apache::Util::escape_uri($uri);
  ```

- **unescape_uri**

This function decodes all %XX hex escape sequences in the given URI.

```
my $unescaped = Apache::Util::unescape_uri($safe_uri);
```

- **unescape_uri_info**

  This function is similar to unescape_uri() but is specialized to remove escape sequences from the query string portion of the URI. The main difference is that it translates the ‘‘+’’ character into spaces as well as recognizing and translating the hex escapes.

  Example:

```
$string = $r->uri->query;
my %data = map { Apache::Util::unescape_uri_info($_) }
             split /[=&]/, $string, -1;
```

  This would correctly translate the query string *"name=Fred+Flintstone&town=Bedrock"* into the hash:

```
data => ’Fred Flintstone’,
town => ’Bedrock’
```

- **parsedate**

  Parses an HTTP date in one of three standard forms:

```
Sun, 06 Nov 1994 08:49:37 GMT  ; RFC 822, updated by RFC 1123

Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036

Sun Nov  6 08:49:37 1994       ; ANSI C’s asctime() format
```

  Example:

```
my $secs = Apache::Util::parsedate($date_str);
```

- **ht_time**

  Format a time string.

  Examples:

```
my $str = Apache::Util::ht_time(time);

my $str = Apache::Util::ht_time(time, "%d %b %Y %T %Z");

my $str = Apache::Util::ht_time(time, "%d %b %Y %T %Z", 0);
```

- **size_string**

  Converts the given file size into a formatted string. The size given in the string will be in units of bytes, kilobytes, or megabytes, depending on the size.

```
my $size = Apache::Util::size_string -s $r->finfo;
```

- **validate_password**

  Validate a plaintext password against a smashed one. Use either crypt() (if available), ap_MD5Encode() or ap_SHA1Encode depending upon the format of the smashed input password.

  Returns true if they match, false otherwise.

```
if (Apache::Util::validate_password("slipknot", "aXYx4GnaCrDQc")) {
    print "password match\n";
}
else {
    print "password mismatch\n";
}
```

8.4 Author

Doug MacEachern

8.5 See Also

perl.

9 Apache::Include - Utilities for mod_perl/mod_include integration

9.1 Synopsis

```
<!--#perl sub="Apache::Include" arg="/perl/ssi.pl" -->
```

9.2 Description

The `Apache::Include` module provides a handler, making it simple to include `Apache::Registry` scripts with the mod_include perl directive.

`Apache::Registry` scripts can also be used in mod_include parsed documents using 'virtual include'.

9.3 Methods

- **Apache::Include->virtual($uri)**

  The `virtual` method may be called to include the output of a given uri in your Perl scripts. Example:

  ```
  use Apache::Include ();

  print "Content-type: text/html\n\n";

  print "before include\n";

  my $uri = "/perl/env.pl";

  Apache::Include->virtual($uri);

  print "after include\n";
  ```

9.4 See Also

perl, mod_perl, mod_include

9.5 Maintainers

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

- **The documentation mailing list**

9.6 Authors

- **Doug MacEachern**

Only the major authors are listed above. For contributors see the Changes file.

10 Apache::Registry - Run unaltered CGI scrips under mod_perl

10.1 Synopsis

```
#in httpd.conf

Alias /perl/ /perl/apache/scripts/ #optional
PerlModule Apache::Registry

<Location /perl>
  SetHandler perl-script
  PerlHandler Apache::Registry
  Options ExecCGI
</Location>
```

10.2 Description

`Apache::Registry` is the Apache module allowing you to run CGI scripts very fast under mod_perl, by compiling all scripts once and then caching them in memory.

URIs in the form of *http://www.example.com/perl/file.pl* will be compiled as the body of a perl subroutine and executed. Each server process or ’child’ will compile the subroutine once and store it in memory. It will recompile it whenever the file is updated on disk. Think of it as an object oriented server with each script implementing a class loaded at runtime.

The file looks much like a "normal" script, but it is compiled or ’evaled’ into a subroutine.

Here’s an example:

```
my $r = Apache->request;
$r->content_type("text/html");
$r->send_http_header;
$r->print("Hi There!");
```

This module emulates the CGI environment, allowing programmers to write scripts that run under CGI or mod_perl without change. Existing CGI scripts may require some changes, simply because a CGI script has a very short lifetime of one HTTP request, allowing you to get away with "quick and dirty" scripting. Using mod_perl and Apache::Registry requires you to be more careful, but it also gives new meaning to the word "quick"!

Be sure to read all mod_perl related documentation for more details, including instructions for setting up an environment that looks exactly like CGI:

```
print "Content-type: text/html\n\n";
print "Hi There!";
```

Note that each httpd process or "child" must compile each script once, so the first request to one server may seem slow, but each request there after will be faster. If your scripts are large and/or make use of many Perl modules, this difference should be noticeable to the human eye.

10.3 Security

`Apache::Registry::handler` will preform the same checks as `mod_cgi` before running the script.

10.4 Environment

The Apache function `exit` overrides the Perl core built-in function.

The environment variable `GATEWAY_INTERFACE` is set to `CGI-Perl/1.1`.

10.5 Command Line Switches on the First Line

Normally when a Perl script is run from the command line or under CGI, arguments on the `#!` line are passed to the perl interpreter for processing.

`Apache::Registry` currently only honors the `-w` switch and will turn on warnings using the `$^W` global variable. Another common switch used with CGI scripts is `-T` to turn on taint checking. This can only be enabled when the server starts with the configuration directive:

```
PerlTaintCheck On
```

However, if taint checking is not enabled, but the `-T` switch is seen, `Apache::Registry` will write a warning to the *error_log*.

10.6 Debugging

You may set the debug level with the `$Apache::Registry::Debug` bitmask

```
1 => log recompile in errorlog
2 => Apache::Debug::dump in case of $@
4 => trace pedantically
```

10.7 Caveats

`Apache::Registry` makes things look just the CGI environment, however, you must understand that this **is not CGI**. Each httpd child will compile your script into memory and keep it there, whereas CGI will run it once, cleaning out the entire process space. Many times you have heard "always use `-w`, always use `-w` and `use strict`". This is more important here than anywhere else!

Your scripts cannot contain the `__END__` or `__DATA__` token to terminate compilation.

10.8 See Also

perl, mod_perl, Apache, Apache::Debug

10.9 Maintainers

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

- **The documentation mailing list**

10.10 Authors

- **Andreas J. Koenig**
- **Doug MacEachern**

Only the major authors are listed above. For contributors see the Changes file.

11 Apache::PerlRun - Run unaltered CGI scripts under mod_perl

11.1 Synopsis

```
#in httpd.conf

Alias /cgi-perl/ /perl/apache/scripts/
PerlModule Apache::PerlRun

<Location /cgi-perl>
  SetHandler perl-script
  PerlHandler Apache::PerlRun
  Options +ExecCGI
  #optional
  PerlSendHeader On
  ...
</Location>
```

11.2 Description

This module's `handler` emulates the CGI environment, allowing programmers to write scripts that run under CGI or mod_perl without change. Unlike `Apache::Registry`, the `Apache::PerlRun` handler does not cache the script inside of a subroutine. Scripts will be "compiled" every request. After the script has run, it's namespace is flushed of all variables and subroutines.

The `Apache::Registry` handler is much faster than `Apache::PerlRun`. However, `Apache::PerlRun` is much faster than CGI as the fork is still avoided and scripts can use modules which have been pre-loaded at server startup time. This module is meant for "Dirty" CGI Perl scripts which relied on the single request lifetime of CGI and cannot run under `Apache::Registry` without cleanup.

11.3 Caveats

If your scripts still have problems running under the `Apache::PerlRun` handler, the `PerlRunOnce` option can be used so that the process running the script will be shutdown. Add this to your httpd.conf:

```
<Location ...>
  PerlSetVar PerlRunOnce On
  ...
</Location>
```

11.4 See Also

perl, mod_perl, Apache::Registry

11.5 Maintainers

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

- **The documentation mailing list**

11.6 Authors

- **Doug MacEachern**

Only the major authors are listed above. For contributors see the Changes file.

12 Apache::RegistryLoader - Compile Apache::Registry scripts at server startup

12.1 Synopsis

```
#in your Perl Startup script:

use Apache::RegistryLoader ();

my $r = Apache::RegistryLoader->new;

$r->handler($uri, $filename);

$r->handler($uri, $filename, $virtual_hostname);
```

12.2 Description

This modules allows compilation of `Apache::Registry` scripts at server startup.

The script's handler routine is compiled by the parent server, of which children get a copy. The `Apache::RegistryLoader` `handler` method takes arguments of `uri` and the `filename`. URI to filename translation normally doesn't happen until HTTP request time, so we're forced to roll our own translation.

If filename is omitted and a `trans` routine was not defined, the loader will try using the *uri* relative to `ServerRoot`. Example:

```
#in httpd.conf
ServerRoot /opt/www/apache
Alias /perl/ /opt/www/apache/perl

#in startup.pl
use Apache::RegistryLoader ();

#/opt/www/apache/perl/test.pl
#is the script loaded from disk here:
Apache::RegistryLoader->new->handler("/perl/test.pl");
```

To make the loader smarter about the uri->filename translation, you may provide the `new` method with a `trans` function to translate the uri to filename.

The following example will pre-load all files ending with `.pl` in the *perl-scripts/* directory relative to `ServerRoot`. The example code assumes the Location URI */perl* is an `Alias` to this directory.

```
{
   use Cwd ();
   use Apache::RegistryLoader ();
   use DirHandle ();
   use strict;

   my $dir = Apache->server_root_relative("perl-scripts/");

   my $rl = Apache::RegistryLoader->new(trans => sub {
       my $uri = shift;
       $uri =~ s:^/perl/:/perl-scripts/:;
```

```
        return Apache->server_root_relative($uri);
    });

    my $dh = DirHandle->new($dir) or die $!;

    for my $file ($dh->read) {
        next unless $file =~ /\.pl$/;
        $rl->handler("/perl/$file");
    }
}
```

12.3 Maintainers

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

- **The documentation mailing list**

12.4 Authors

- **Doug MacEachern**
- **Stas Bekman (Rewrote the `handler()` to report and handle all the possible erroneous conditions).**

Only the major authors are listed above. For contributors see the Changes file.

12.5 See Also

Apache::Registry, Apache, mod_perl

13 Apache::StatINC - Reload %INC files when updated on disk

13.1 Synopsis

```
#httpd.conf or some such
#can be any Perl*Handler
PerlInitHandler Apache::StatINC
```

13.2 Description

When Perl pulls a file via `require`, it stores the filename in the global hash `%INC`. The next time Perl tries to `require` the same file, it sees the file in `%INC` and does not reload from disk. This module's handler iterates over `%INC` and reloads the file if it has changed on disk.

Note that StatINC operates on the current context of `@INC`. Which means, when called as a Perl*Handler it will not see `@INC` paths added or removed by Apache::Registry scripts, as the value of `@INC` is saved on server startup and restored to that value after each request. In other words, if you want StatINC to work with modules that live in custom `@INC` paths, you should modify `@INC` when the server is started. Besides, `use lib` in startup scripts, you can also set the `PERL5LIB` variable in the httpd's environment to include any non-standard 'lib' directories that you choose. For example, you might use a script called 'start_httpd' to start apache, and include a line like this:

```
PERL5LIB=/usr/local/foo/myperllibs; export PERL5LIB
```

When you have problems with modules not being reloaded, please refer to the following lines in *perlmodlib*:

"Always use `-w`. Try to `use strict;` (or `use strict qw(...);`). Remember that you can add `no strict qw(...);` to individual blocks of code that need less strictness. Always use `-w`. Always use `-w`! Follow the guidelines in the perlstyle(1) manual."

Warnings when running under mod_perl is enabled with `PerlWarn On` in your httpd.conf.

It will most likely help you to find the problem. Really.

13.3 Options

- **StatINC_UndefOnReload**

  Normally, `StatINC` will turn of warnings to avoid "Subroutine redefined" warnings when it reloads a file. However, this does not disable the Perl mandatory warning when re-defining `constant` subroutines (see perldoc perlsub). With this option On, StatINC will invoke the `Apache::Symbol undef_functions` method to avoid these mandatory warnings:

  ```
  PerlSetVar StatINC_UndefOnReload On
  ```

- **StatINC_Debug**

You can make `StatINC` tell when it reloads a module by setting this option to on.

```
PerlSetVar StatINC_Debug 1
```

The only used debug level is currently 1.

13.4 SEE ALSO

mod_perl

13.5 Maintainers

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

- **Ask Bjoern Hanse <ask (at) netcetera.dk>**
- **The documentation mailing list**

13.6 Authors

- **Doug MacEachern**
- **Ask Bjoern Hansen**

Only the major authors are listed above. For contributors see the Changes file.

14 Apache::test - Facilitates testing of Apache::* modules

14.1 Synopsis

```
# In Makefile.PL
use Apache::test;
my %params = Apache::test->get_test_params();
Apache::test->write_httpd_conf(%params, include => $more_directives);
*MY::test = sub { Apache::test->MM_test(%params) };

# In t/*.t script (or test.pl)
use Apache::test qw(skip_test have_httpd);
skip_test unless have_httpd;
(Some more methods of Doug’s that I haven’t reviewed or documented yet)
```

14.2 Description

This module helps authors of `Apache::*` modules write test suites that can query an actual running Apache server with mod_perl and their modules loaded into it.

Its functionality is generally separated into methods that go in a *Makefile.PL* to configure, start, and stop the server, and methods that go in one of the test scripts to make HTTP queries and manage the results.

14.3 Methods

14.3.1 get_test_params()

This will ask the user a few questions about where the httpd binary is, and what user/group/port should be used when running the server. It will return a hash of the information it discovers. This hash is suitable for passing to the `write_httpd_conf()` method.

14.3.2 write_httpd_conf(%params)

This will write a basic *httpd.conf* file suitable for starting a HTTP server during the `make test` stage. A hash of key/value pairs that affect the written file can be passed as arguments. The following keys are recognized:

- **conf_file**

  The path to the file that will be created. Default is *t/httpd.conf*.

- **port**

  The port that the Apache server will listen on.

- **user**

  The user that the Apache server will run as.

- **group**

  The group that the Apache server will run as.

- **include**

  Any additional text you want added at the end of the config file. Typically you'll have some `PerlModule` and `Perl*Handler` directives to pass control to the module you're testing. The `blib/` directories will be added to the `@INC` path when searching for modules, so that's nice.

14.3.3 MM_test(%params)

This method helps write a Makefile that supports running a web server during the `make test` stage. When you execute `make test`, `make` will run `make start_httpd`, `make run_tests`, and `make kill_httpd` in sequence. You can also run these commands independently if you want.

Pass the hash of parameters returned by `get_test_params()` as an argument to `MM_test()`.

To patch into the `ExtUtils::MakeMaker` wizardry (voodoo?), typically you'll do the following in your *Makefile.PL*:

```
*MY::test = sub { Apache::test->MM_test(%params) };
```

14.3.4 fetch

```
Apache::test->fetch($request);
Apache::test->fetch($user_agent, $request);
```

Call this method in a test script in order to fetch a page from the running web server. If you pass two arguments, the first should be an `LWP::UserAgent` object, and the second should specify the request to make of the server. If you only pass one argument, it specifies the request to make.

The request can be specified either by a simple string indicating the URI to fetch, or by a hash reference, which gives you more control over the request. The following keys are recognized in the hash:

- **uri**

  The URI to fetch from the server. If the URI does not begin with `http`, we prepend `http://localhost:$PORT` so that we make requests of the test server.

- **method**

  The request method to use. Default is `GET`.

- **content**

  The request content body. Typically used to simulate HTML fill-out form submission for `POST` requests. Default is null.

- **headers**

  A hash of headers you want sent with the request. You might use this to send cookies or provide some application-specific header.

If you don't provide a `headers` parameter and you set the `method` to `POST`, then we assume that you're trying to simulate HTML form submission and we add a `Content-Type` header with a value of `application/x-www-form-urlencoded`.

In a scalar context, `fetch()` returns the content of the web server's response. In a list context, `fetch()` returns the content and the `HTTP::Response` object itself. This can be handy if you need to check the response headers, or the HTTP return code, or whatever.

14.3.5 static_modules

```
Example: $mods = Apache::test->static_modules('/path/to/httpd');
```

This method returns a hashref whose keys are all the modules statically compiled into the given httpd binary. The corresponding values are all 1.

14.4 Examples

No good examples yet. Example submissions are welcome. In the meantime, see http://forum.swarthmore.edu/~ken/modules/Apache-AuthCookie/ , which I'm retrofitting to use `Apache::test`.

14.5 To Do

The `MM_test` method doesn't try to be very smart, it just writes the text that seems to work in my configuration. I am morally against using the `make` command for installing Perl modules (though of course I do it anyway), so I haven't looked into this very much. Send bug reports or better (patches).

I've got lots of code in my `Apache::AuthCookie` module (etc.) that assists in actually making the queries of the running server. I plan to add that to this module, but first I need to compare what's already here that does the same stuff.

14.6 Kudos

To Doug MacEachern for writing the first version of this module.

To caelum@debian.org (Rafael Kitover) for contributing the code to parse existing httpd.conf files for `--enable-shared=max` and DSOs.

14.7 Caveats

Except for making sure that the mod_perl distribution itself can run `make test` okay, I haven't tried very hard to keep compatibility with older versions of this module. In particular `MM_test()` has changed and probably isn't usable in the old ways, since some of its assumptions are gone. But none of this was ever documented, and `MM_test()` doesn't seem to actually be used anywhere in the mod_perl disribution, so I don't feel so bad about it.

14.8 Maintainers

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

- **The documentation mailing list**

14.9 Authors

- **Doug MacEachern**
- **Ken Williams**

Only the major authors are listed above. For contributors see the Changes file.

15 Apache::Symdump - Symbol table snapshots

15.1 Synopsis

```
PerlLogHandler Apache::Symdump
```

15.2 Description

`Apache::Symdump` will record snapshots of the Perl symbol table for you to look at later.

It records them in `ServerRoot/logs/symdump.$$.$n`. Where `$$` is the process id and `$n` is incremented each time the handler is run. The `diff` utility can be used to compare snapshots and get an idea of what might be making a process grow. Normally, new symbols come from modules or scripts that were not preloaded, the Perl method cache, etc.

```
% diff -u symdump.$$.0 symdump.$$.1
```

15.3 Caveats

`Apache::Symdump` does not cleanup up its snapshot files, do so simply by:

```
% rm logs/symdump.* logs/incdump.*
```

15.4 See Also

`Devel::Symdump`, Apache::Leak

15.5 Maintainers

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

- **The documentation mailing list**

15.6 Authors

- **Doug MacEachern**

Only the major authors are listed above. For contributors see the Changes file.

16 Apache::src - Methods for locating and parsing bits of Apache source code

16.1 Synopsis

```
use Apache::src ();
my $src = Apache::src->new;
```

16.2 Description

This module provides methods for locating and parsing bits of Apache source code.

16.3 Methods

- **new**

  Create an object blessed into the **Apache::src** class.

  ```
  my $src = Apache::src->new;
  ```

- **dir**

  Top level directory where source files are located.

  ```
  my $dir = $src->dir;
  -d $dir or die "can't stat $dir $!\n";
  ```

- **main**

  Apache's source tree was reorganized during development of version 1.3. So, common header files such as `httpd.h` are in different directories between versions less than 1.3 and those equal to or greater. This method will return the right directory.

  Example:

  ```
  -e join "/", $src->main, "httpd.h" or die "can't stat httpd.h\n";
  ```

- **find**

  Searches for apache source directories, return a list of those found.

  Example:

  ```
  for my $dir ($src->find) {
     my $yn = prompt "Configure with $dir ?", "y";
     ...
  }
  ```

- **inc**

  Print include paths for MakeMaker's **INC** argument to `WriteMakefile`.

Example:

```
use ExtUtils::MakeMaker;

use Apache::src ();

WriteMakefile(
    'NAME'    => 'Apache::Module',
    'VERSION' => '0.01',
    'INC'     => Apache::src->new->inc,
);
```

- **module_magic_number**

  Return the **MODULE_MAGIC_NUMBER** defined in the apache source.

  Example:

  ```
  my $mmn = $src->module_magic_number;
  ```

- **httpd_version**

  Return the server version.

  Example:

  ```
  my $v = $src->httpd_version;
  ```

- **otherldflags**

  Return other ld flags for MakeMaker's **dynamic_lib** argument to `WriteMakefile`. This might be needed on systems like AIX that need special flags to the linker to be able to reference mod_perl or httpd symbols.

  Example:

  ```
  use ExtUtils::MakeMaker;

  use Apache::src ();

  WriteMakefile(
      'NAME'        => 'Apache::Module',
      'VERSION'     => '0.01',
      'INC'         => Apache::src->new->inc,
      'dynamic_lib' => {
  'OTHERLDFLAGS' => Apache::src->new->otherldflags,
      },
  );
  ```

16.4 Author

Doug MacEachern

17 Apache::Leak - Module for tracking memory leaks in mod_perl code

17.1 Synopsis

```
use Apache::Leak;

leak_test {
    my $obj = Foo->new;
    $obj->thingy;
};
#now look in error_log for results
```

17.2 Description

`Apache::Leak` is a module built to track memory leaks in mod_perl code.

17.3 See Also

`Devel::Leak`

17.4 Author

Doug MacEachern

Leak.xs was derived from Nick Ing-Simmons' Devel::Leak

18 Apache::FakeRequest - fake request object for debugging

18.1 Synopsis

```
use Apache::FakeRequest;
my $request = Apache::FakeRequest->new(method_name => 'value', ...);
```

18.2 Description

`Apache::FakeRequest` is used to set up an empty Apache request object that can be used for debugging.

The `Apache::FakeRequest` methods just set internal variables of the same name as the method and return the value of the internal variables. Initial values for methods can be specified when the object is created. The `print` method prints to STDOUT.

Subroutines for Apache constants are also defined so that using `Apache::Constants` while debugging works, although the values of the constants are hard-coded rather than extracted from the Apache source code.

```
#!/usr/bin/perl

use Apache::FakeRequest ();
use mymodule ();

my $request = Apache::FakeRequest->new('get_remote_host'=>'foobar.com');
mymodule::handler($request);
```

18.3 Authors

Doug MacEachern, with contributions from Andrew Ford <A.Ford@ford-mason.co.uk>.

19 Apache::Debug - Utilities for debugging embedded perl code

19.1 Synopsis

```
use Apache::Debug ();

Apache::Debug::dump($r, SERVER_ERROR, "Uh Oh!");
```

19.2 Description

This module sends what may be helpful debugging info to the client rather that the error log.

19.3 Maintainers

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

The mod_perl docs list.

19.4 Authors

- **Rob Hartill**

Only the major authors are listed above. For contributors see the Changes file.

20 Apache::Symbol - Things for symbol things

20.1 Synopsis

```
use Apache::Symbol ();

@ISA = qw(Apache::Symbol);
```

20.2 Description

`Apache::Symbol` helps mod_perl users avoid Perl warnings related with redefined constant functions.

`perlsub/Constant Functions` says:

```
If you redefine a subroutine which was eligible for inlining you'll
get a mandatory warning.  (You can use this warning to tell whether
or not a particular subroutine is considered constant.)  The warning
is considered severe enough not to be optional because previously
compiled invocations of the function will still be using the old
value of the function.
```

mandatory warning means there is **no** way to avoid this warning no matter what tricks you pull in Perl. This is bogus for us mod_perl users when restarting the server with `PerlFreshRestart` on or when `Apache::StatINC` pulls in a module that has changed on disk.

You can, however, pull some tricks with XS to avoid this warning,
`Apache::Symbol::undef_functions` does just that.

20.3 Arguments

`undef_functions` takes two arguments: `skip` and `only_undef_exports`.

`skip` is a regular expression indicating the function names to skip.

Use the `only_undef_exports` flag to undef only those functions which are listed in `@EXPORT`, `@EXPORT_OK`, `%EXPORT_TAGS`, or `@EXPORT_EXTRAS`. `@EXPORT_EXTRAS` is not used by the Exporter, it is only exists to communicate with `undef_functions`.

As a special case, if none of the `EXPORT` variables are defined ignore `only_undef_exports`. This takes care of trivial modules that don't use the Exporter.

20.4 Players

This module and the undefining of functions is optional, if you wish to have this functionality enabled, there are one or more switches you need to know about.

- **PerlRestartHandler**

`Apache::Symbol` defines a `PerlRestartHandler` which can be useful in conjuction with `PerlFreshRestart On` as it will avoid subroutine redefinition messages. Configure like so:

```
PerlRestartHandler Apache::Symbol
```

- **Apache::Registry**

  By placing the *SYNOPSIS* bit in you script, `Apache::Registry` will undefine subroutines in your script before it is re-compiled to avoid "subroutine re-defined" warnings.

- **Apache::StatINC**

  See `Apache::StatINC`'s docs.

- **APACHE_SYMBOL_UNIVERSAL**

  If this environment variable is true when Symbol.pm is compiled, it will define `UNIVERSAL::undef_functions`, which means all classes will inherit **Apache::Symbol::undef_functions**.

- **Others**

  Modules such as `HTML::Embperl` and `Apache::ePerl` which compile and script cache scripts ala `Apache::Registry` style can use `undef_functions` with this bit of code:

  ```
  if($package->can('undef_functions')) {
      $package->undef_functions;
  }
  ```

  Where `$package` is the name of the package in which the script is being re-compiled.

20.5 See Also

perlsub, `Devel::Symdump`

20.6 Maintainers

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

- **The documentation mailing list**

20.7 Authors

- **Doug MacEachern**

Only the major authors are listed above. For contributors see the Changes file.

21 Apache::SIG - Override apache signal handlers with Perl’s

21.1 Synopsis

```
PerlFixupHandler Apache::SIG
```

21.2 Description

When a client drops a connection and apache is in the middle of a write, a timeout will occur and httpd sends a `SIGPIPE`. When apache's `SIGPIPE` handler is used, Perl may be left in the middle of it's eval context, causing bizarre errors during subsequent requests are handled by that child. When `Apache::SIG` is used, it installs a different `SIGPIPE` handler which rewinds the context to make sure Perl is back to normal state, preventing these bizarre errors.

If you would like to log when a request was cancelled by a `SIGPIPE` in your Apache *access_log*, you can declare `Apache::SIG` as a handler (any `Perl*Handler` will do, as long as it is run before `PerlHandler`, e.g. `PerlFixupHandler`), and you must also define a custom `LogFormat` in your httpd.conf, like so:

```
PerlFixupHandler Apache::SIG
LogFormat "%h %l %u %t \"%r\" %s %b %{SIGPIPE}e"
```

If the server has noticed that the request was cancelled via a `SIGPIPE`, then the log line will end with `1`, otherwise it will just be a dash.

21.3 Caveats

The signal handler in this package uses the subprocess_env table of the main request object to supply the `SIGPIPE` "environment variable" to the log handler. If you already use the key `SIGPIPE` in your `subprocess_env` table, then you can redefine the key like this:

```
$Apache::SIG::PipeKey = 'my_SIGPIPE';
```

and log it like this:

```
LogFormat "%h %l %u %t \"%r\" %s %b %{my_SIGPIPE}e"
```

21.4 Maintainers

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

- **The documentation mailing list**

21.5 Authors

- **Doug MacEachern**
- **Doug Bagley**

Only the major authors are listed above. For contributors see the Changes file.

21.6 See Also

perlvar(1)

22 Apache::PerlSections - Utilities for work with Perl sections

22.1 Synopsis

```
use Apache::PerlSections ();
```

22.2 Description

It is possible to configure you server entirely in Perl using `<Perl>` sections in *httpd.conf*. This module is here to help you with such a task.

22.3 Methods

- **dump**

  This method will dump out all the configuration variables mod_perl will be feeding to the apache config gears. The output is suitable to read back in via `eval`.

  Example:

```
<Perl>

use Apache::PerlSections ();

$Port = 8529;

$Location{"/perl"} = {
   SetHandler => "perl-script",
   PerlHandler => "Apache::Registry",
   Options => "ExecCGI",
};

@DirectoryIndex = qw(index.htm index.html);

$VirtualHost{"www.foo.com"} = {
   DocumentRoot => "/tmp/docs",
   ErrorLog => "/dev/null",
   Location => {
     "/" => {
       Allowoverride => 'All',
       Order => 'deny,allow',
       Deny  => 'from all',
       Allow => 'from foo.com',
     },
   },
};

print Apache::PerlSections->dump;

</Perl>
```

This will print something like this:

```
package Apache::ReadConfig;
#scalars:

$Port = 8529;

#arrays:

@DirectoryIndex = (
  'index.htm',
  'index.html'
);

#hashes:

%Location = (
  '/perl' => {
    PerlHandler => 'Apache::Registry',
    SetHandler => 'perl-script',
    Options => 'ExecCGI'
  }
);

%VirtualHost = (
  'www.foo.com' => {
    Location => {
      '/' => {
        Deny => 'from all',
        Order => 'deny,allow',
        Allow => 'from foo.com',
        Allowoverride => 'All'
      }
    },
    DocumentRoot => '/tmp/docs',
    ErrorLog => '/dev/null'
  }
);

1;
__END__
```

- **store**

  This method will call the `dump` method, writing the output to a file, suitable to be pulled in via `require`.

  Example:

  ```
  Apache::PerlSections->store("httpd_config.pl");

  require 'httpd_config.pl';
  ```

22.4 See Also

mod_perl, `Data::Dumper`, `Devel::Symdump`

22.5 Maintainers

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

- **The documentation mailing list**

22.6 Authors

- **Doug MacEachern**

Only the major authors are listed above. For contributors see the Changes file.

23 Apache::httpd_conf - Generate an httpd.conf file

23.1 Synopsis

```
use Apache::httpd_conf ();

Apache::httpd_conf->write(Port => 8888);
```

23.2 Description

The Apache::httpd_conf module will generate a tiny httpd.conf file, which pulls itself back in via a `<Perl>` section.

Any additional arguments passed to the `write` method will be added to the generated httpd.conf file, and will override those defaults set in the `<Perl>` section. This module is handy mostly for starting httpd servers to test mod_perl scripts and modules.

23.3 Author

Doug MacEachern

23.4 See Also

mod_perl, Apache::PerlSections

24 Apache::Status - Embedded interpreter status information

24.1 Synopsis

```
<Location /perl-status>

  # disallow public access
  Order Deny, Allow
  Deny from all
  Allow from 127.0.0.1

  SetHandler  perl-script
  PerlHandler Apache::Status
</Location>
```

24.2 Description

The `Apache::Status` module provides some information about the status of the Perl interpreter embedded in the server.

Configure like so:

```
<Location /perl-status>

  # disallow public access
  Order Deny, Allow
  Deny from all
  Allow from 127.0.0.1

  SetHandler  perl-script
  PerlHandler Apache::Status
</Location>
```

Other modules can "plugin" a menu item like so:

```
Apache::Status->menu_item(
  'DBI' => "DBI connections", #item for Apache::DBI module
   sub {
       my ($r,$q) = @_; #request and CGI objects
       my (@strings);
       push @strings,  "blobs of html";
       return \@strings;     #return an array ref
   }
) if Apache->module("Apache::Status"); #only if Apache::Status is loaded
```

WARNING: `Apache::Status` must be loaded before these modules via the `PerlModule` or `Perl-Require` directives.

24.3 Options

- **StatusOptionsAll**

This single directive will enable all of the options described below.

```
PerlSetVar StatusOptionsAll On
```

- **StatusDumper**

  When browsing symbol tables, the values of arrays, hashes and scalars can be viewed via `Data::Dumper` if this configuration variable is set to On:

  ```
  PerlSetVar StatusDumper On
  ```

- **StatusPeek**

  With this option *On* and the `Apache::Peek` module installed, functions and variables can be viewed ala `Devel::Peek` style:

  ```
  PerlSetVar StatusPeek On
  ```

- **StatusLexInfo**

  With this option On and the `B::LexInfo` module installed, subroutine lexical variable information can be viewed.

  ```
  PerlSetVar StatusLexInfo On
  ```

- **StatusDeparse**

  With this option On and `B::Deparse` version 0.59 or higher (included in Perl 5.005_59+), subroutines can be "deparsed".

  ```
  PerlSetVar StatusDeparse On
  ```

  Options can be passed to `B::Deparse::new` like so:

  ```
  PerlSetVar StatusDeparseOptions "-p -sC"
  ```

  See the `B::Deparse` manpage for details.

- **StatusTerse**

  With this option *On*, text-based op tree graphs of subroutines can be displayed, thanks to `B::Terse`.

  ```
  PerlSetVar StatusTerse On
  ```

- **StatusTerseSize**

  With this option On and the `B::TerseSize` module installed, text-based op tree graphs of subroutines and their size can be displayed. See the `B::TerseSize` docs for more info.

  ```
  PerlSetVar StatusTerseSize On
  ```

- **StatusTerseSizeMainSummary**

With this option On and the `B::TerseSize` module installed, a "Memory Usage" will be added to the `Apache::Status` main menu. This option is disabled by default, as it can be rather cpu intensive to summarize memory usage for the entire server. It is strongly suggested that this option only be used with a development server running in `-X` mode, as the results will be cached.

```
PerlSetVar StatusTerseSizeMainSummary On
```

- **StatusGraph**

  When `StatusDumper` is enabled, another link "OP Tree Graph" will be present with the dump if this configuration variable is set to On:

  ```
  PerlSetVar StatusGraph
  ```

  This requires the `B` module (part of the Perl compiler kit) and `B::Graph` (version 0.03 or higher) module to be installed along with the `dot` program.

  Dot is part of the graph visualization toolkit from AT&T:
  http://www.research.att.com/sw/tools/graphviz/).

  WARNING: Some graphs may produce very large images, some graphs may produce no image if `B::Graph`'s output is incorrect.

- **Dot**

  Location of the dot program for `StatusGraph`, if other than */usr/bin* or */usr/local/bin*.

- **GraphDir**

  Directory where `StatusGraph` should write it's temporary image files. Default is *$Server-Root/logs/b_graphs*.

24.4 Prerequisites

The `Devel::Symdump` module, version **2.00** or higher.

24.5 See Also

perl, Apache, `Devel::Symdump`, `Data::Dumper`, `B`, `B::Graph`

24.6 Maintainers

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

- **The documentation mailing list**

24.7 Authors

- **Doug MacEachern**

Only the major authors are listed above. For contributors see the Changes file.

25 Apache::Resource - Limit resources used by httpd children

25.1 Synopsis

```
PerlModule Apache::Resource
#set child memory limit in megabytes
#default is 64 Meg
PerlSetEnv PERL_RLIMIT_DATA 32:48

#linux does not honor RLIMIT_DATA
#RLIMIT_AS (address space) will work to limit the size of a process
PerlSetEnv PERL_RLIMIT_AS 32:48

#set child cpu limit in seconds
#default is 360 seconds
PerlSetEnv PERL_RLIMIT_CPU 120

PerlChildInitHandler Apache::Resource
```

25.2 Description

`Apache::Resource` uses the `BSD::Resource` module, which uses the C function `setrlimit` to set limits on system resources such as memory and cpu usage.

Any `RLIMIT` operation available to limit on your system can be set by defining that operation as an environment variable with a `PERL_` prefix. See your system `setrlimit` manpage for available resources which can be limited.

The following limit values are in megabytes: `DATA`, `RSS`, `STACK`, `FSIZE`, `CORE`, `MEMLOCK`; all others are treated as their natural unit.

If the value of the variable is of the form `S:H`, `S` is treated as the soft limit, and `H` is the hard limit. If it is just a single number, it is used for both soft and hard limits.

25.3 Defaults

To set reasonable defaults for all RLIMITs, add this to your httpd.conf:

```
PerlSetEnv PERL_RLIMIT_DEFAULTS On
PerlModule Apache::Resource
```

25.4 Maintainers

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

- **The documentation mailing list**

25.5 Authors

- **Doug MacEachern**

Only the major authors are listed above. For contributors see the Changes file.

25.6 SEE ALSO

`BSD::Resource`, `setrlimit(2)`

26 Apache::SizeLimit - Because size does matter.

26.1 Synopsis

This module allows you to kill off Apache httpd processes if they grow too large. You can choose to set up the process size limiter to check the process size on every request:

```
# in your startup.pl:
use Apache::SizeLimit;
# sizes are in KB
$Apache::SizeLimit::MAX_PROCESS_SIZE  = 10000; # 10MB
$Apache::SizeLimit::MIN_SHARE_SIZE    = 1000;  # 1MB
$Apache::SizeLimit::MAX_UNSHARED_SIZE = 12000; # 12MB

# in your httpd.conf:
PerlFixupHandler Apache::SizeLimit
# you can set this up as any Perl*Handler that handles part of the
# request, even the LogHandler will do.
```

Or you can just check those requests that are likely to get big, such as CGI requests. This way of checking is also easier for those who are mostly just running CGI.pm/Registry scripts:

```
# in your CGI:
use Apache::SizeLimit;
&Apache::SizeLimit::setmax(10000);        # Max size in KB
&Apache::SizeLimit::setmin(1000);        # Min share in KB
&Apache::SizeLimit::setmax_unshared(12000); # Max unshared size in KB
```

Since checking the process size can take a few system calls on some platforms (e.g. linux), you may want to only check the process size every *N* times. To do so, put this in your *startup.pl* or CGI:

```
$Apache::SizeLimit::CHECK_EVERY_N_REQUESTS = 2;
```

This will only check the process size every other time the process size checker is called.

26.2 Description

This module allows you to kill off Apache httpd processes if they grow too large.

This module is highly platform dependent, please read the Caveats section.

This module was written in response to questions on the mod_perl mailing list on how to tell the httpd process to exit if it gets too big.

Actually there are two big reasons your httpd children will grow. First, it could have a bug that causes the process to increase in size dramatically, until your system starts swapping. Second, your process just does stuff that requires a lot of memory, and the more different kinds of requests your server handles, the larger the httpd processes grow over time.

This module will not really help you with the first problem. For that you should probably look into `Apache::Resource` or some other means of setting a limit on the data size of your program. BSD-ish systems have `setrlimit()` which will croak your memory gobbling processes. However it is a little

violent, terminating your process in mid-request.

This module attempts to solve the second situation where your process slowly grows over time. The idea is to check the memory usage after every request, and if it exceeds a threshold, exit gracefully.

By using this module, you should be able to discontinue using the Apache configuration directive `MaxRequestsPerChild`, although for some folks, using both in combination does the job. Personally, I just use the technique shown in this module and set my `MaxRequestsPerChild` value to 6000.

26.3 Shared Memory Options

In addition to simply checking the total size of a process, this module can factor in how much of the memory used by the process is actually being shared by copy-on-write. If you don't understand how memory is shared in this way, take a look at the Sharing Memory section.

You can take advantage of the shared memory information by setting a minimum shared size and/or a maximum unshared size. Experience on one heavily trafficked mod_perl site showed that setting maximum unshared size and leaving the others unset is the most effective policy. This is because it only kills off processes that are truly using too much physical RAM, allowing most processes to live longer and reducing the process churn rate.

26.4 Caveats

This module is platform dependent, since finding the size of a process is pretty different from OS to OS, and some platforms may not be supported. In particular, the limits on minimum shared memory and maximum shared memory are currently only supported on Linux and BSD. If you can contribute support for another OS, please do.

Currently supported OSes:

- **linux**

  For linux we read the process size out of */proc/self/status*. This is a little slow, but usually not too bad. If you are worried about performance, try only setting up the the exit handler inside CGIs (with the `setmax` function), and see if the `CHECK_EVERY_N_REQUESTS` option is of benefit.

- **solaris 2.6 and above**

  For solaris we simply retrieve the size of */proc/self/as*, which contains the address-space image of the process, and convert to KB. Shared memory calculations are not supported.

  NOTE: This is only known to work for solaris 2.6 and above. Evidently the */proc* filesystem has changed between 2.5.1 and 2.6. Can anyone confirm or deny?

- ***bsd***

Uses `BSD::Resource::getrusage()` to determine process size. This is pretty efficient (a lot more efficient than reading it from the */proc* fs anyway).

- **AIX?**

  Uses `BSD::Resource::getrusage()` to determine process size. Not sure if the shared memory calculations will work or not. AIX users?

If your platform is not supported, and if you can tell me how to check for the size of a process under your OS (in KB), then I will add it to the list. The more portable/efficient the solution, the better, of course.

26.5 Todo

Possibly provide a perl make/install so that the SizeLimit.pm is created at build time with only the code you need on your platform.

If Apache was started in non-forking mode, should hitting the size limit cause the process to exit?

26.6 Maintainers

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

- **The documentation mailing list**

26.7 Authors

- Doug Bagley <doug+modperl (at) bagley.org>, channeling Procrustes.
- Brian Moseley <ix (at) maz.org>: Solaris 2.6 support
- Doug Steinwand and Perrin Harkins <perrin (at) elem.com>: added support for shared memory and additional diagnostic info

Only the major authors are listed above. For contributors see the Changes file.

Table of Contents: