

mod_perl APIs

The Apache::, APR:: and ModPerl:: namespaces APIs for
mod_perl 2.0

Last modified Fri Jul 30 11:00:01 2004 GMT

Part I: Apache:: Core API

- 1. Apache -- A ghost mod_perl 2.0 class
There is no Apache class per se.
- 2. Apache::Access - A Perl API for Apache request object
Apache::Access provides the Perl API for Apache request object.
- 3. Apache::CmdParms - Perl API for XXX
META: to be completed
- 4. Apache::Command - Perl API for XXX
META: to be completed
- 5. Apache::compat -- 1.0 backward compatibility functions deprecated in 2.0
Apache::compat provides mod_perl 1.0 compatibility layer and can be used to smooth the transition process to mod_perl 2.0.
- 6. Apache::Connection - Perl API for Apache connection object
META: to be completed
- 7. Apache::Const - Perl Interface for Apache Constants
- 8. Apache::Directive - Perl API for manipulating Apache configuration tree
Apache::Directive allows its users to search and navigate the internal Apache configuration.
- 9. Apache::Filter - Perl API for Apache 2.0 Filtering
Apache::Filter provides the Perl API for Apache 2.0 filtering framework.
- 10. Apache::FilterRec - Perl API for manipulating the Apache filter record
META: to be completed
- 11. Apache::HookRun - Perl API for XXX
META: to be completed
- 12. Apache::Log - Perl API for Apache Logging Methods
Apache::Log provides the Perl API for Apache logging methods.
- 13. Apache::Module - Perl API for creating and working with Apache modules
META: to be completed
- 14. Apache::PerlSections - Default Handler for Perl sections
With <Perl >...</Perl> sections, it is possible to configure your server entirely in Perl.
- 15. Apache::Process - Perl API for XXX
META: to be completed

- 16. Apache::RequestIO - Perl API for Apache request record IO
Apache::RequestIO provides the API to perform IO on the *Apache request object*.
- 17. Apache::RequestRec - Perl API for Apache request record accessors
Apache::RequestRec provides the Perl API for Apache request object.
- 18. Apache::RequestUtil - Perl API for Apache request record utils
META: to be completed
- 19. Apache::Response - Perl API for Apache HTTP request response methods
META: to be completed
- 20. Apache::Server - Perl API for for Apache server record accessors
META: to be completed
- 21. Apache::ServerUtil - Perl API for XXX
Apache::ServerUtil provides the Perl API for Apache server object.
- 22. Apache::SubProcess -- Executing SubProcesses from mod_perl
Apache::SubProcess provides the Perl API for running and communicating with processes spawned from mod_perl handlers.
- 23. Apache::SubRequest - Perl API for Apache subrequests
META: to be completed
- 24. Apache::URI - Perl API for manipulating URIs
META: to be completed
- 25. Apache::Util - Perl API for XXX
META: to be completed

Part II: APR:: Core API

- 26. APR - Perl Interface for libapr and libaprutil Libraries
Notes on how to use APR outside mod_perl 2.0.
- 27. APR::Base64 - Perl API for XXX
META: to be completed
- 28. APR::Brigade - Perl API for XXX
META: to be completed
- 29. APR::Bucket - Perl API for XXX
META: to be completed
- 30. APR::Const - Perl Interface for APR Constants

- 31. APR::Date - Perl API for XXX
META: to be completed
- 32. APR::Finfo - Perl API for XXX
META: to be completed
- 33. APR::NetLib - Perl API for XXX
META: to be completed
- 34. APR::PerlIO -- An APR Perl IO layer
APR::PerlIO implements a Perl IO layer using APR's file manipulation as its internals.
- 35. APR::Pool - Perl API for XXX
META: to be completed
- 36. APR::SockAddr - Perl API for XXX
META: to be completed
- 37. APR::Socket - Perl API for XXX
META: to be completed
- 38. APR::Table - Perl API for for manipulating opaque string-content table
APR::Table allows its users to manipulate opaque string-content tables.
- 39. APR::ThreadMutex - Perl API for XXX
META: to be completed
- 40. APR::URI - Perl API for XXX
META: to be completed
- 41. APR::Util - Perl API for XXX
META: to be completed

Part III: ModPerl::

- 42. ModPerl::MethodLookup -- Map mod_perl 2.0 modules, objects and methods
mod_perl 2.0 provides many methods, which reside in various modules. One has to load each of the modules before using the desired methods. ModPerl::MethodLookup provides the Perl API for finding module names which contain methods in question and other helper functions, like figuring out what methods defined by some module, or what methods can be called on a given object.
- 43. ModPerl::MM -- A "subclass" of ExtUtils::MakeMaker for mod_perl 2.0
ModPerl::MM is a "subclass" of ExtUtils::MakeMaker for mod_perl 2.0, to a degree of sub-classability of ExtUtils::MakeMaker.
- 44. ModPerl::PerlRun - Run unaltered CGI scripts under mod_perl

- 45. **ModPerl::Registry** - Run unaltered CGI scripts persistently under mod_perl
URIs in the form of `http://example.com/perl/test.pl` will be compiled as the body of a Perl subroutine and executed. Each child process will compile the subroutine once and store it in memory. It will recompile it whenever the file (e.g. *test.pl* in our example) is updated on disk. Think of it as an object oriented server with each script implementing a class loaded at runtime.
- 46. **ModPerl::RegistryBB** - Run unaltered CGI scripts persistently under mod_perl
`ModPerl::RegistryBB` is similar to `ModPerl::Registry`, but does the bare minimum (mnemonic: BB = Bare Bones) to compile a script file once and run it many times, in order to get the maximum performance. Whereas `ModPerl::Registry` does various checks, which add a slight overhead to response times.
- 47. **ModPerl::RegistryCooker** - Cook mod_perl 2.0 Registry Modules
`ModPerl::RegistryCooker` is used to create flexible and overridable registry modules which emulate `mod_cgi` for Perl scripts. The concepts are discussed in the manpage of the following modules: `ModPerl::Registry`, `ModPerl::Registry` and `ModPerl::RegistryBB`.
- 48. **ModPerl::RegistryLoader** - Compile `ModPerl::RegistryCooker` scripts at server startup
This modules allows compilation of scripts, running under packages derived from `ModPerl::RegistryCooker`, at server startup. The script's handler routine is compiled by the parent server, of which children get a copy and thus saves some memory by initially sharing the compiled copy with the parent and saving the overhead of script's compilation on the first request in every httpd instance.
- 49. **ModPerl::Util** - Helper mod_perl 2.0 Functions
`ModPerl::Util` provides mod_perl 2.0 util functions.

Part IV: Helper Modules / Applications

- 50. **Apache::porting** -- a helper module for mod_perl 1.0 to mod_perl 2.0 porting
`Apache::porting` helps to port mod_perl 1.0 code to run under mod_perl 2.0. It doesn't provide any back-compatibility functionality, however it knows trap calls to methods that are no longer in the mod_perl 2.0 API and tell what should be used instead if at all. If you attempts to use mod_perl 2.0 methods without first loading the modules that contain them, it will tell you which modules you need to load. Finally if your code tries to load modules that no longer exist in mod_perl 2.0 it'll also tell you what are the modules that should be used instead.
- 51. **Apache::Reload** - Reload Perl Modules when Changed on Disk
`Apache::Reload` reloads modules that change on the disk.
- 52. **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: Internal Modules

Table of Contents:

- 53. `ModPerl::BuildMM` -- A "subclass" of `ModPerl::MM` used for building `mod_perl 2.0`
`ModPerl::BuildMM` is a "subclass" of `ModPerl::MM` used for building `mod_perl 2.0`. Refer to *`ModPerl::MM`* manpage.

See search.cpan.org or www.perldoc.com for documentation of the 3rd party Apache : : modules.

1 Apache -- A ghost mod_perl 2.0 class

1.1 Synopsis

1.2 Description

There is no Apache class per se.

There are several modules that put their functions into the `Apache::` namespace. For example `ModPerl::Util` defines a function `Apache::current_callback()`:

```
use ModPerl::Util;
my $callback = Apache::current_callback();
```

There are several modules that require the *Apache* class as the first argument to the class methods that they define. For example `Apache::Server` defines a class method `Apache->server`:

```
use Apache::Server;
my $server = Apache->server;
```

There are several modules that install constants into the `Apache::` namespace. For example `Apache::ServerUtil` defines a constant `Apache::Server::server_root`:

```
use Apache::ServerUtil;
my $server_root = Apache::Server::server_root;
```

To use this functions and methods you need to load the module that defines them. If you aren't sure which module contains the symbol you are after, use the helper module `ModPerl::MethodLookup`.

1.3 See Also

`mod_perl 2.0` documentation.

1.4 Copyright

`mod_perl 2.0` and its core modules are copyrighted under The Apache Software License, Version 1.1.

1.5 Authors

The `mod_perl` development team and numerous contributors.

2 Apache::Access - A Perl API for Apache request object

2.1 Synopsis

META: needs to be completed

```
use Apache::Access ( );
```

2.2 Description

Apache::Access provides the Perl API for Apache request object.

2.3 API

Apache::Access provides the following functions and/or methods:

2.3.1 *allow_options*

META: Autogenerated - needs to be reviewed/completed

Retrieve the value of Options for this request

```
$ret = $r->allow_options();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

the Options bitmask

2.3.2 *allow_overrides*

META: Autogenerated - needs to be reviewed/completed

Retrieve the value of the AllowOverride for this request

```
$ret = $r->allow_overrides();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

the overrides bitmask

2.3.3 *get_remote_logname*

META: Autogenerated - needs to be reviewed/completed

Retrieve the login name of the remote user. Undef if it could not be determined

```
$ret = $r->get_remote_logname();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (string)**

The user logged in to the client machine

2.3.4 *note_auth_failure*

META: Autogenerated - needs to be reviewed/completed

Setup the output headers so that the client knows how to authenticate itself the next time, if an authentication request failed. This function works for both basic and digest authentication

```
$r->note_auth_failure();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: no return value**

2.3.5 *note_basic_auth_failure*

META: Autogenerated - needs to be reviewed/completed

Setup the output headers so that the client knows how to authenticate itself the next time, if an authentication request failed. This function works only for basic authentication

```
$r->note_basic_auth_failure();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: no return value**

2.3.6 *note_digest_auth_failure*

META: Autogenerated - needs to be reviewed/completed

Setup the output headers so that the client knows how to authenticate itself the next time, if an authentication request failed. This function works only for digest authentication

```
$r->note_digest_auth_failure();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: no return value**

2.3.7 *satisfies*

META: Autogenerated - needs to be reviewed/completed

How the requires lines must be met.

```
$ret = $r->satisfies();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

How the requirements must be met. One of:

```
Apache::SATISFY_ANY    -- any of the requirements must be met.
Apache::SATISFY_ALL    -- all of the requirements must be met.
Apache::SATISFY_NOSPEC -- There are no applicable satisfy lines
```

2.3.8 *some_auth_required*

META: Autogenerated - needs to be reviewed/completed

Can be used within any handler to determine if any authentication is required for the current request

```
$ret = $r->some_auth_required();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

1 if authentication is required, 0 otherwise

2.4 See Also

mod_perl 2.0 documentation.

2.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

2.6 Authors

The mod_perl development team and numerous contributors.

3 Apache::CmdParms - Perl API for XXX

3.1 Synopsis

```
use Apache::CmdParms ();
```

META: to be completed

3.2 Description

META: to be completed

3.3 API

Apache::CmdParms provides the following functions and/or methods:

3.3.1 *info*

META: Autogenerated - needs to be reviewed/completed

Argument to command from cmd_table

```
$obj->info($newval);
```

- **arg1:** \$obj (Apache::CmdParms)
- **arg2:** \$newval (XXX)
- **ret:** no return value

3.3.2 *override*

META: Autogenerated - needs to be reviewed/completed

Which allow-override bits are set

```
$obj->override($newval);
```

- **arg1:** \$obj (Apache::CmdParms)
- **arg2:** \$newval (integer)
- **ret:** no return value

3.3.3 *limited*

META: Autogenerated - needs to be reviewed/completed

Which methods are <Limit>ed

3.3.4 *limited_xmethods*

```
$obj->limited($newval);
```

- **arg1:** `$obj` (`Apache::CmdParms`)
- **arg2:** `$newval` (number)
- **ret:** no return value

3.3.4 *limited_xmethods*

META: Autogenerated - needs to be reviewed/completed

methods which are limited

```
$obj->limited_xmethods($newval);
```

- **arg1:** `$obj` (`Apache::CmdParms`)
- **arg2:** `$newval` (`APR::ArrayHeader`)
- **ret:** no return value

3.3.5 *xlimited*

META: Autogenerated - needs to be reviewed/completed

methods which are xlimited

```
$obj->xlimited($newval);
```

- **arg1:** `$obj` (`Apache::CmdParms`)
- **arg2:** `$newval` (`Apache::MethodList`)
- **ret:** no return value

3.3.6 *config_file*

META: Autogenerated - needs to be reviewed/completed

Config file structure.

```
$obj->config_file($newval);
```

- **arg1:** `$obj` (`Apache::CmdParms`)
- **arg2:** `$newval` (`Apache::ConfigFile`)
- **ret:** no return value

3.3.7 *directive*

META: Autogenerated - needs to be reviewed/completed

the directive specifying this command

```
$obj->directive($newval);
```

- **arg1:** `$obj` (`Apache::CmdParms`)
- **arg2:** `$newval` (`Apache::Directive`)
- **ret:** no return value

3.3.8 *pool*

META: Autogenerated - needs to be reviewed/completed

Pool to allocate new storage in

```
$obj->pool($newval);
```

- **arg1:** `$obj` (`Apache::CmdParms`)
- **arg2:** `$newval` (`APR::Pool`)
- **ret:** no return value

3.3.9 *temp_pool*

META: Autogenerated - needs to be reviewed/completed

Pool for scratch memory; persists during configuration, but wiped before the first request is served...

```
$obj->temp_pool($p);
```

- **arg1:** `$obj` (`Apache::CmdParms`)
- **arg2:** `$p` (`APR::Pool`)
- **ret:** no return value

3.3.10 *server*

META: Autogenerated - needs to be reviewed/completed

server_rec being configured for

```
$obj->server($s);
```

- **arg1:** `$obj` (`Apache::CmdParms`)
- **arg2:** `$s` (`Apache::Server`)
- **ret:** no return value

3.3.11 *path*

META: Autogenerated - needs to be reviewed/completed

If configuring for a directory, pathname of that directory. NOPE! That's what it meant previous to the existence of <Files>, <Location> and regex matching. Now the only usefulness that can be derived from this field is whether a command is being called in a server context (path == NULL) or being called in a dir context (path != NULL).

```
$obj->path($newval);
```

- **arg1:** \$obj (Apache::CmdParms)
- **arg2:** \$newval (string)
- **ret:** no return value

3.3.12 *cmd*

META: Autogenerated - needs to be reviewed/completed

configuration command

```
$obj->cmd($newval);
```

- **arg1:** \$obj (Apache::CmdParms)
- **arg2:** \$newval (Apache::Command)
- **ret:** no return value

3.3.13 *context*

META: Autogenerated - needs to be reviewed/completed

per_dir_config vector passed to handle_command

```
$obj->context($newval);
```

- **arg1:** \$obj (Apache::CmdParms)
- **arg2:** \$newval (Apache::ConfVector)
- **ret:** no return value

3.3.14 *err_directive*

META: Autogenerated - needs to be reviewed/completed

directive with syntax error

```
$obj->err_directive($newval);
```

- **arg1:** `$obj` (**Apache::CmdParms**)
- **arg2:** `$newval` (**Apache::Directive**)
- **ret:** no return value

3.4 See Also

mod_perl 2.0 documentation.

3.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

3.6 Authors

The mod_perl development team and numerous contributors.

4 Apache::Command - Perl API for XXX

4.1 Synopsis

```
use Apache::Command ();
```

META: to be completed

4.2 Description

META: to be completed

4.3 API

Apache::Command provides the following functions and/or methods:

4.3.1 *check_cmd_context*

META: Autogenerated - needs to be reviewed/completed

```
$ret = $cmd->check_cmd_context($forbidden);
```

- **arg1: \$cmd (Apache::CmdParms)**

The command to check

- **arg2: \$forbidden (XXX)**

Where the command is forbidden.

- **ret: \$ret (string)**

Error string on error, NULL on success

4.3.2 *soak_end_container*

META: Autogenerated - needs to be reviewed/completed

Read all data between the current <foo> and the matching </foo>. All of this data is forgotten immediately.

```
$ret = $cmd->soak_end_container($directive);
```

- **arg1: \$cmd (Apache::CmdParms)**

The cmd_parms to pass to the directives inside the container

- **arg2: \$directive (string)**

4.3.3 next

The directive name to read until

- **ret: \$ret (string)**

Error string on failure, NULL on success

4.3.3 next

META: Autogenerated - needs to be reviewed/completed

```
$ret = $cmd->next();
```

- **arg1: \$cmd (Apache::Command)**
- **ret: \$ret (Apache::Command)**

4.3.4 name

META: Autogenerated - needs to be reviewed/completed

Name of this command

```
$obj->name($newval);
```

- **arg1: \$obj (Apache::Command)**
- **arg2: \$newval (string)**
- **ret: no return value**

4.3.5 cmd_data

META: Autogenerated - needs to be reviewed/completed

Extra data, for functions which implement multiple commands...

```
$obj->cmd_data($newval);
```

- **arg1: \$obj (Apache::Command)**
- **arg2: \$newval (XXX)**
- **ret: no return value**

4.3.6 req_override

META: Autogenerated - needs to be reviewed/completed

What overrides need to be allowed to enable this command.

```
$obj->req_override($newval);
```

- **arg1:** `$obj` (**Apache::Command**)
- **arg2:** `$newval` (integer)
- **ret:** no return value

4.3.7 *args_how*

META: Autogenerated - needs to be reviewed/completed

What the command expects as arguments

```
$obj->args_how($newval);
```

- **arg1:** `$obj` (**Apache::Command**)
- **arg2:** `$newval` (integer)
- **ret:** no return value

4.3.8 *errmsg*

META: Autogenerated - needs to be reviewed/completed

'usage' message, in case of syntax errors

```
$obj->errmsg($newval);
```

- **arg1:** `$obj` (**Apache::Command**)
- **arg2:** `$newval` (string)
- **ret:** no return value

4.4 See Also

mod_perl 2.0 documentation.

4.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

4.6 Authors

The mod_perl development team and numerous contributors.

5 Apache::compat -- 1.0 backward compatibility functions deprecated in 2.0

5.1 Synopsis

```
# either add at the very beginning of startup.pl
use Apache2
use Apache::compat;
# or httpd.conf
PerlModule Apache2
PerlModule Apache::compat

# override and restore compat functions colliding with mp2 API
Apache::compat::override_mp2_api('Apache::Connection::local_addr');
my ($local_port, $local_addr) = sockaddr_in($c->local_addr);
Apache::compat::restore_mp2_api('Apache::Connection::local_addr');
```

5.2 Description

Apache::compat provides mod_perl 1.0 compatibility layer and can be used to smooth the transition process to mod_perl 2.0.

It includes functions that have changed their API or were removed in mod_perl 2.0. If your code uses any of those functions, you should load this module at the server startup, and everything should work as it did in 1.0. If it doesn't please report the bug, but before you do that please make sure that your code does work properly under mod_perl 1.0.

However, remember, that it's implemented in pure Perl and not C, therefore its functionality is not optimized and it's the best to try to port your code not to use deprecated functions and stop using the compatibility layer.

5.3 Compatibility Functions Colliding with mod_perl 2.0 API

Most of the functions provided by Apache::compat don't interfere with mod_perl 2.0 API. However there are several functions which have the same name in the mod_perl 1.0 and mod_perl 2.0 API, accept the same number of arguments, but either the arguments themselves aren't the same or the return values are different. For example the mod_perl 1.0 code:

```
require Socket;
my $sockaddr_in = $c->local_addr;
my ($local_port, $local_addr) = Socket::sockaddr_in($sockaddr_in);
```

should be adjusted to be:

```
require Apache::Connection;
require APR::SocketAddr;
my $sockaddr = $c->local_addr;
my ($local_port, $local_addr) = ($sockaddr->port, $sockaddr->ip_get);
```

to work under mod_perl 2.0.

As you can see in mod_perl 1.0 API `local_addr()` was returning a `SOCKADDR_IN` object (see the `Socket` perl manpage), in mod_perl 2.0 API it returns an `APR::SocketAddr` object, which is a totally different beast. If `Apache::compat` overrides the function `local_addr()` to be back-compatible with mod_perl 1.0 API. Any code that relies on this function to work as it should under mod_perl 2.0 will be broken. Therefore the solution is not to override `local_addr()` by default. Instead a special API is provided which overrides colliding functions only when needed and which can be restored when no longer needed. So for example if you have code from mod_perl 1.0:

```
my ($local_port, $local_addr) = Socket::sockaddr_in($c->local_addr);
```

and you aren't ready to port it to to use the mp2 API:

```
my ($local_port, $local_addr) = ($c->local_addr->port,
                                  $c->local_addr->ip_get);
```

you could do the following:

```
Apache::compat::override_mp2_api('Apache::Connection::local_addr');
my ($local_port, $local_addr) = Socket::sockaddr_in($c->local_addr);
Apache::compat::restore_mp2_api('Apache::Connection::local_addr');
```

Notice that you need to restore the API as soon as possible.

Both `override_mp2_api()` and `restore_mp2_api()` accept a list of functions to operate on.

5.3.1 Available Overridable Functions

At the moment the following colliding functions are available for overriding:

- `Apache::RequestRec::notes`
- `Apache::RequestRec::finfo`
- `Apache::Connection::local_addr`
- `Apache::Connection::remote_addr`
- `Apache::server_root_relative`

5.4 Use in CPAN Modules

The short answer: **Do not use** `Apache::compat` in CPAN modules.

The long answer:

`Apache::compat` is useful during the mod_perl 1.0 code porting. Though remember that it's implemented in pure Perl. In certain cases it overrides mod_perl 2.0 methods, because their API is very different and doesn't map 1:1 to mod_perl 1.0. So if anything, not under user's control, loads `Apache::compat` user's code is forced to use the potentially slower method. Which is quite bad.

Some users may choose to keep using `Apache::compat` in production and it may perform just fine. Other users will choose not to use that module, by porting their code to use `mod_perl 2.0` API. However it should be users' choice whether to load this module or not and not to be enforced by CPAN modules.

If you port your CPAN modules to work with `mod_perl 2.0`, you should follow the porting Perl and XS module guidelines.

Users that are stuck with CPAN modules preloading `Apache::compat`, can prevent this from happening by adding

```
$INC{'Apache/compat.pm'} = __FILE__;
```

at the very beginning of their *startup.pl*. But this will most certainly break the module that needed this module.

5.5 API

You should be reading the `mod_perl 1.0` API docs for usage of the methods and functions in this package, since what this module is doing is providing a backwards compatibility and it makes no sense to duplicate documentation.

Another important document to read is: *Migrating from mod_perl 1.0 to mod_perl 2.0* which covers all `mod_perl 1.0` constants, functions and methods that have changed in `mod_perl 2.0`.

5.6 See Also

`mod_perl 2.0` documentation.

5.7 Copyright

`mod_perl 2.0` and its core modules are copyrighted under The Apache Software License, Version 1.1.

5.8 Authors

The `mod_perl` development team and numerous contributors.

6 Apache::Connection - Perl API for Apache connection object

6.1 Synopsis

```
use Apache::Connection ();
```

META: to be completed

6.2 Description

META: to be completed

6.3 API

Apache::Connection provides the following functions and/or methods:

6.3.1 *aborted*

Check whether the connection is still open

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

- **arg1: \$c (Apache::Connection)**
- **ret: \$status (number)**

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

6.3.2 *base_server*

Physical vhost this connection came in on

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

- **arg1: \$c (Apache::Connection)**
- **ret: \$base_server (Apache::Server)**

6.3.3 *bucket_alloc*

META: Autogenerated - needs to be reviewed/completed

The bucket allocator to use for all bucket/brigade creations

```
$ba = $c->bucket_alloc();
```

- **arg1: \$c (Apache::Connection)**
- **ret: \$ba (APR::BucketAlloc)**

6.3.4 conn_config

META: Autogenerated - needs to be reviewed/completed

Notes on **this** connection

```
$ret = $c->conn_config();
```

- **arg1: \$c (Apache::Connection)**
- **ret: \$ret (Apache::ConfVector)**

6.3.5 id

ID of this connection; unique at any point in time

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

- **arg1: \$c (Apache::Connection)**
- **ret: \$id (integer)**

6.3.6 input_filters

A list of input filters to be used for this connection

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

- **arg1: \$c (Apache::Connection)**
- **ret: \$input_filters (Apache::Filter)**

The first filter in the connection input filters chain.

6.3.7 keepalive

Are we going to keep the connection alive for another request?

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

- **arg1: \$c (Apache::Connection)**
- **ret: \$status (integer)**

6.3.8 local_addr

Get this connection's local socket address

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

- **arg1:** `$c` (**Apache::Connection**)
- **ret:** `$sa` (**APR::SockAddr**)

6.3.9 *local_host*

used for `ap_get_server_name` when `UseCanonicalName` is set to `DNS` (ignores setting of `Host-nameLookups`)

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

- **arg1:** `$c` (**Apache::Connection**)
- **ret:** `$local_host` (**string**)

6.3.10 *local_ip*

server IP address

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

- **arg1:** `$c` (**Apache::Connection**)
- **ret:** `$local_ip` (**string**)

6.3.11 *notes*

META: Autogenerated - needs to be reviewed/completed

send note from one module to another, must remain valid for all requests on this conn

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

- **arg1:** `$c` (**Apache::Connection**)
- **arg2:** `$notes` (**APR::Table**)
- **ret:**

6.3.12 *output_filters*

META: Autogenerated - needs to be reviewed/completed

A list of output filters to be used for this connection

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

- **arg1:** `$c` (**Apache::Connection**)
- **ret:** `$output_filters` (**Apache::Filter**)

The first filter in the connection output filters chain.

6.3.13 *pool*

Pool associated with this connection

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

- **arg1:** `$c` (**Apache::Connection**)
- **ret:** `$p` (**APR::Pool**)

6.3.14 *remote_addr*

Get this connection's remote socket address

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

- **arg1:** `$c` (**Apache::Connection**)
- **ret:** `$sa` (**APR::SockAddr**)

6.3.15 *remote_ip*

Client's IP address

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

- **arg1:** `$c` (**Apache::Connection**)
- **ret:** `$remote_ip` (**string**)

6.3.16 *remote_host*

Client's DNS name, if known. NULL if DNS hasn't been checked, "" if it has and no address was found. N.B. Only access this through `get_remote_host()`

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

- **arg1:** `$c` (**Apache::Connection**)
- **ret:** `$remote_host` (**string**)

6.3.17 *remote_logname*

Only ever set if doing rfc1413 lookups. N.B. Only access this through `get_remote_logname()`

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


- **arg1: \$c (Apache::Connection)**
- **ret: \$remote_logname (string)**

6.3.18 *sbh*

META: Autogenerated - needs to be reviewed/completed

handle to scoreboard information for this connection

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

- **arg1: \$c (Apache::Connection)**
- **ret: \$sbh (XXX)**

6.4 See Also

mod_perl 2.0 documentation.

6.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

6.6 Authors

The mod_perl development team and numerous contributors.

7 Apache::Const - Perl Interface for Apache Constants

7.1 Synopsis

```
use Apache::Const -compile => qw(constant names ...);
```

7.2 Constants

7.2.1 *:cmd_how*

```
use Apache::Const -compile => qw(:cmd_how);
```

The `:cmd_how` group is for XXX constants.

7.2.1.1 `Apache::FLAG`

7.2.1.2 `Apache::ITERATE`

7.2.1.3 `Apache::ITERATE2`

7.2.1.4 `Apache::NO_ARGS`

7.2.1.5 `Apache::RAW_ARGS`

7.2.1.6 `Apache::TAKE1`

7.2.1.7 `Apache::TAKE12`

7.2.1.8 `Apache::TAKE123`

7.2.1.9 `Apache::TAKE13`

7.2.1.10 `Apache::TAKE2`

7.2.1.11 `Apache::TAKE23`

7.2.1.12 `Apache::TAKE3`

7.2.2 *:common*

```
use Apache::Const -compile => qw(:common);
```

The `:common` group is for XXX constants.

7.2.3 :config

7.2.2.1 Apache::AUTH_REQUIRED

7.2.2.2 Apache::DECLINED

7.2.2.3 Apache::DONE

7.2.2.4 Apache::FORBIDDEN

7.2.2.5 Apache::NOT_FOUND

7.2.2.6 Apache::OK

7.2.2.7 Apache::REDIRECT

7.2.2.8 Apache::SERVER_ERROR

7.2.3 :config

```
use Apache::Const -compile => qw(:config);
```

The :config group is for XXX constants.

7.2.3.1 Apache::DECLINE_CMD

7.2.4 :filter_type

```
use Apache::Const -compile => qw(:filter_type);
```

The :filter_type group is for XXX constants.

7.2.4.1 Apache::FTYPE_CONNECTION

7.2.4.2 Apache::FTYPE_CONTENT_SET

7.2.4.3 Apache::FTYPE_NETWORK

7.2.4.4 Apache::FTYPE_PROTOCOL

7.2.4.5 Apache::FTYPE_RESOURCE

7.2.4.6 Apache::FTYPE_TRANSCODE

7.2.5 :http

```
use Apache::Const -compile => qw(:http);
```

The :http group is for XXX constants.

7.2.5.1 Apache::HTTP_ACCEPTED

7.2.5.2 Apache::HTTP_BAD_GATEWAY

7.2.5.3 Apache::HTTP_BAD_REQUEST

7.2.5.4 Apache::HTTP_CONFLICT

7.2.5.5 Apache::HTTP_CONTINUE

7.2.5.6 Apache::HTTP_CREATED

7.2.5.7 Apache::HTTP_EXPECTATION_FAILED

7.2.5.8 Apache::HTTP_FAILED_DEPENDENCY

7.2.5.9 Apache::HTTP_FORBIDDEN

7.2.5.10 Apache::HTTP_GATEWAY_TIME_OUT

7.2.5.11 Apache::HTTP_GONE

7.2.5.12 Apache::HTTP_INSUFFICIENT_STORAGE

7.2.5.13 Apache::HTTP_INTERNAL_SERVER_ERROR

7.2.5.14 Apache::HTTP_LENGTH_REQUIRED

7.2.5.15 Apache::HTTP_LOCKED

7.2.5.16 Apache::HTTP_METHOD_NOT_ALLOWED

7.2.5.17 Apache::HTTP_MOVED_PERMANENTLY

7.2.5.18 Apache::HTTP_MOVED_TEMPORARILY

7.2.5.19 Apache::HTTP_MULTIPLE_CHOICES

- 7.2.5.20 Apache::HTTP_MULTI_STATUS
- 7.2.5.21 Apache::HTTP_NON_AUTHORITATIVE
- 7.2.5.22 Apache::HTTP_NOT_ACCEPTABLE
- 7.2.5.23 Apache::HTTP_NOT_EXTENDED
- 7.2.5.24 Apache::HTTP_NOT_FOUND
- 7.2.5.25 Apache::HTTP_NOT_IMPLEMENTED
- 7.2.5.26 Apache::HTTP_NOT_MODIFIED
- 7.2.5.27 Apache::HTTP_NO_CONTENT
- 7.2.5.28 Apache::HTTP_OK
- 7.2.5.29 Apache::HTTP_PARTIAL_CONTENT
- 7.2.5.30 Apache::HTTP_PAYMENT_REQUIRED
- 7.2.5.31 Apache::HTTP_PRECONDITION_FAILED
- 7.2.5.32 Apache::HTTP_PROCESSING
- 7.2.5.33 Apache::HTTP_PROXY_AUTHENTICATION_REQUIRED
- 7.2.5.34 Apache::HTTP_RANGE_NOT_SATISFIABLE
- 7.2.5.35 Apache::HTTP_REQUEST_ENTITY_TOO_LARGE
- 7.2.5.36 Apache::HTTP_REQUEST_TIME_OUT
- 7.2.5.37 Apache::HTTP_REQUEST_URI_TOO_LARGE
- 7.2.5.38 Apache::HTTP_RESET_CONTENT
- 7.2.5.39 Apache::HTTP_SEE_OTHER
- 7.2.5.40 Apache::HTTP_SERVICE_UNAVAILABLE

7.2.5.41 Apache::HTTP_SWITCHING_PROTOCOLS

7.2.5.42 Apache::HTTP_TEMPORARY_REDIRECT

7.2.5.43 Apache::HTTP_UNAUTHORIZED

7.2.5.44 Apache::HTTP_UNPROCESSABLE_ENTITY

7.2.5.45 Apache::HTTP_UNSUPPORTED_MEDIA_TYPE

7.2.5.46 Apache::HTTP_USE_PROXY

7.2.5.47 Apache::HTTP_VARIANT_ALSO_VARIES

7.2.6 :input_mode

```
use Apache::Const -compile => qw(:input_mode);
```

The :input_mode group is for XXX constants.

7.2.6.1 Apache::MODE_EATCRLF

7.2.6.2 Apache::MODE_EXHAUSTIVE

7.2.6.3 Apache::MODE_GETLINE

7.2.6.4 Apache::MODE_INIT

7.2.6.5 Apache::MODE_READBYTES

7.2.6.6 Apache::MODE_SPECULATIVE

7.2.7 :log

```
use Apache::Const -compile => qw(:log);
```

The :log group is for XXX constants.

7.2.7.1 Apache::LOG_ALERT

7.2.7.2 Apache::LOG_CRIT

7.2.7.3 Apache::LOG_DEBUG

7.2.7.4 Apache::LOG_EMERG

7.2.7.5 Apache::LOG_ERR

7.2.7.6 Apache::LOG_INFO

7.2.7.7 Apache::LOG_LEVELMASK

7.2.7.8 Apache::LOG_NOTICE

7.2.7.9 Apache::LOG_STARTUP

7.2.7.10 Apache::LOG_TOCLIENT

7.2.7.11 Apache::LOG_WARNING

7.2.8 :methods

```
use Apache::Const -compile => qw(:methods);
```

The :methods group is for XXX constants.

7.2.8.1 Apache::METHODS

7.2.8.2 Apache::M_BASELINE_CONTROL

7.2.8.3 Apache::M_CHECKIN

7.2.8.4 Apache::M_CHECKOUT

7.2.8.5 Apache::M_CONNECT

7.2.8.6 Apache::M_COPY

7.2.8.7 Apache::M_DELETE

7.2.8.8 Apache::M_GET

7.2.8.9 Apache::M_INVALID

7.2.8.10 Apache::M_LABEL

7.2.8.11 Apache::M_LOCK

7.2.8.12 Apache::M_MERGE

7.2.8.13 Apache::M_MKACTIVITY

7.2.8.14 Apache::M_MKCOL

7.2.8.15 Apache::M_MKWORKSPACE

7.2.8.16 Apache::M_MOVE

7.2.8.17 Apache::M_OPTIONS

7.2.8.18 Apache::M_PATCH

7.2.8.19 Apache::M_POST

7.2.8.20 Apache::M_PROPFIND

7.2.8.21 Apache::M_PROPPATCH

7.2.8.22 Apache::M_PUT

7.2.8.23 Apache::M_REPORT

7.2.8.24 Apache::M_TRACE

7.2.8.25 Apache::M_UNCHECKOUT

7.2.8.26 Apache::M_UNLOCK

7.2.8.27 Apache::M_UPDATE

7.2.8.28 Apache::M_VERSION_CONTROL

7.2.9 :mpmq

```
use Apache::Const -compile => qw(:mpmq);
```

The :mpmq group is for querying MPM properties.

7.2.9.1 Apache::MPMQ_NOT_SUPPORTED

7.2.9.2 Apache::MPMQ_STATIC

7.2.9.3 Apache::MPMQ_DYNAMIC

7.2.9.4 Apache::MPMQ_MAX_DAEMON_USED

7.2.9.5 Apache::MPMQ_IS_THREADED

7.2.9.6 Apache::MPMQ_IS_FORKED

7.2.9.7 Apache::MPMQ_HARD_LIMIT_DAEMONS

7.2.9.8 Apache::MPMQ_HARD_LIMIT_THREADS

7.2.9.9 Apache::MPMQ_MAX_THREADS

7.2.9.10 Apache::MPMQ_MIN_SPARE_DAEMONS

7.2.9.11 Apache::MPMQ_MIN_SPARE_THREADS

7.2.9.12 Apache::MPMQ_MAX_SPARE_DAEMONS

7.2.9.13 Apache::MPMQ_MAX_SPARE_THREADS

7.2.9.14 Apache::MPMQ_MAX_REQUESTS_DAEMON

7.2.9.15 Apache::MPMQ_MAX_DAEMONS

7.2.10 :options

```
use Apache::Const -compile => qw(:options);
```

The :options group is for XXX constants.

7.2.10.1 Apache::OPT_ALL

7.2.10.2 Apache::OPT_EXECCGI

7.2.10.3 Apache::OPT_INCLUDES

7.2.10.4 Apache::OPT_INCNOEXEC

7.2.10.5 Apache::OPT_INDEXES

7.2.10.6 Apache::OPT_MULTI

7.2.10.7 Apache::OPT_NONE

7.2.10.8 Apache::OPT_SYM_LINKS

7.2.10.9 Apache::OPT_SYM_OWNER

7.2.10.10 Apache::OPT_UNSET

7.2.11 :override

```
use Apache::Const -compile => qw(:override);
```

The :override group is for XXX constants.

7.2.11.1 Apache::ACCESS_CONF

7.2.11.2 Apache::OR_ALL

7.2.11.3 Apache::OR_AUTHCFG

7.2.11.4 Apache::OR_FILEINFO

7.2.11.5 Apache::OR_INDEXES

7.2.11.6 Apache::OR_LIMIT

7.2.11.7 Apache::OR_NONE

7.2.11.8 Apache::OR_OPTIONS

7.2.11.9 Apache::OR_UNSET

7.2.11.10 Apache::RSRC_CONF

7.2.12 :platform

```
use Apache::Const -compile => qw(:platform);
```

The :platform group is for constants that may differ from OS to OS.

7.3 See Also

7.2.12.1 **Apache::CRLF**

7.2.12.2 **Apache::CR**

7.2.12.3 **Apache::LF**

7.2.13 ***:remotehost***

```
use Apache::Const -compile => qw(:remotehost);
```

The `:remotehost` group is for XXX constants.

7.2.13.1 **Apache::REMOTE_DOUBLE_REV**

7.2.13.2 **Apache::REMOTE_HOST**

7.2.13.3 **Apache::REMOTE_NAME**

7.2.13.4 **Apache::REMOTE_NOLOOKUP**

7.2.14 ***:satisfy***

```
use Apache::Const -compile => qw(:satisfy);
```

The `:satisfy` group is for XXX constants.

7.2.14.1 **Apache::SATISFY_ALL**

7.2.14.2 **Apache::SATISFY_ANY**

7.2.14.3 **Apache::SATISFY_NOSPEC**

7.2.15 ***:types***

```
use Apache::Const -compile => qw(:types);
```

The `:types` group is for XXX constants.

7.2.15.1 **Apache::DIR_MAGIC_TYPE**

7.3 See Also

`mod_perl 2.0 documentation`.

7.4 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

7.5 Authors

The mod_perl development team and numerous contributors.

8 Apache::Directive - Perl API for manipulating Apache configuration tree

8.1 Synopsis

```
use Apache::Directive ();

my $tree = Apache::Directive->conftree;

my $documentroot = $tree->lookup('DocumentRoot');

my $vhost = $tree->lookup('VirtualHost', 'localhost:8000');
my $servername = $vhost->{'ServerName'};

print $tree->as_string;

use Data::Dumper;
print Dumper($tree->as_hash);

my $node = $tree;
while ($node) {

    #do something with $node

    if (my $kid = $node->first_child) {
        $node = $kid;
    }
    elsif (my $next = $node->next) {
        $node = $next;
    }
    else {
        if (my $parent = $node->parent) {
            $node = $parent->next;
        }
        else {
            $node = undef;
        }
    }
}
```

8.2 Description

`Apache::Directive` allows its users to search and navigate the internal Apache configuration.

Internally, this information is stored in a tree structure. Each node in the tree has a reference to its parent (if it's not the root), its first child (if any), and to its next sibling.

8.3 Class Methods

8.3.1 *conftree()*

Returns the root of the configuration tree.

```
$tree = Apache::Directive->conftree();
```

- **arg1: *Apache::Directive* (class name)**
- **ret: *\$tree* (*Apache::Directive*)**

8.4 Object Methods

8.4.1 *as_hash()*

Returns a hash representation of the configuration tree, in a format suitable for inclusion in the <Perl> sections.

```
$config_hash = $conftree->as_hash();
```

- **arg1: *\$conftree* (*Apache::Directive*)**

The config tree to stringify

- **ret: *\$config_hash* (HASH)**

8.4.2 *as_string()*

Returns a string representation of the configuration tree, in httpd.conf format.

```
$string = $conftree->as_string();
```

- **arg1: *\$conftree* (*Apache::Directive*)**

The config tree to stringify

- **ret: *\$string* (string)**

8.4.3 *lookup()*

Returns node(s) matching a certain value.

```
$node = $conftree->lookup($directive, $args);
@nodes = $conftree->lookup($directive, $args);
```

- **arg1: *\$conftree* (*Apache::Directive*)**

The config tree to stringify

- **arg2: \$directive (string)**
- **opt arg3: args (string)**
- **ret: \$string (string)**

In list context, it will return all matching nodes.

In scalar context, it will return only the first matching node.

If called with only \$directive value, this will return all nodes from that directive. For example:

```
@Alias = $conftree->lookup('Alias');
```

would return all nodes for Alias directives.

If called with an extra \$args argument, this will return only nodes where both the directive and the args matched. For example:

```
$VHost = $tree->lookup('VirtualHost', '_default_:8000');
```

If called with only one \$directive value, this will return all nodes from that directive:

```
@Alias = $tree->lookup('Alias');
```

Would return all nodes for Alias directives.

If called with an extra \$args argument, this will return only nodes where both the directive and the args matched:

```
$VHost = $tree->lookup('VirtualHosts', '_default_:8000');
```

8.4.4 walk_config

META: Autogenerated - needs to be reviewed/completed

Walk a config tree and setup the server's internal structures

```
$ret = $conftree->walk_config($cmdparms, $section_vector);
```

- **arg1: \$conftree (Apache::Directive)**

The config tree to walk

- **arg2: \$cmdparms (Apache::CmdParms)**

The cmd_parms to pass to all functions

- **arg3: \$section_vector (Apache::ConfVector)**

The per-section config vector.

- **ret: \$ret (string)**

Error string on error, undef otherwise

8.4.5 *directive*

Returns the name of the directive in \$node.

```
$name = $node->directive();
```

- **arg1: \$node (Apache::Directive)**
- **ret: \$directive (string)**

8.4.6 *args*

The arguments for the current directive, stored as a space separated list

```
$args = $node->args();
```

- **arg1: \$node (Apache::Directive)**
- **ret: \$args (string)**

8.4.7 *next*

The next directive node in the tree

```
$next_node = $node->next();
```

- **arg1: \$node (Apache::Directive)**
- **ret: \$next_node (Apache::Directive)**

Returns the next sibling of \$node, undef if there is none

8.4.8 *first_child*

The first child node of this directive

```
$subtree = $node->first_child;
```

- **arg1: \$node (Apache::Directive)**
- **ret: \$child_node (Apache::Directive)**

Returns the first child node of \$node, undef if there is none

8.4.9 *parent*

META: Autogenerated - needs to be reviewed/completed

The parent node of this directive

```
$parent_node = $node->parent();
```

- **arg1:** `$node` (**Apache::Directive**)
- **ret:** `parent_node` (**Apache::Directive**)

Returns the parent of `$node`, undef if this node is the root node

8.4.10 *data*

META: Autogenerated - needs to be reviewed/completed

directive's module can store add'l data here

```
$ret = $conftree->data($newval);
```

- **arg1:** `$conftree` (**Apache::Directive**)
- **arg2:** `$newval XXX`
- **ret:** `XXX`

8.4.11 *filename*

Returns the filename the configuration node was created from

```
$filename = $node->filename();
```

- **arg1:** `$node` (**Apache::Directive**)
- **ret:** `$filename` (**string**)

8.4.12 *line_num*

Returns the line number in `filename` this node was created from

```
$lineno = $node->line_num();
```

- **arg1:** `$node` (**Apache::Directive**)
- **arg2:** `$lineno` (**integer**)

8.5 See Also

8.5 See Also

`mod_perl` 2.0 documentation.

8.6 Copyright

`mod_perl` 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

8.7 Authors

The `mod_perl` development team and numerous contributors.

9 Apache::Filter - Perl API for Apache 2.0 Filtering

9.1 Synopsis

```
use Apache::Filter ();
```

META: to be completed

9.2 Description

`Apache::Filter` provides the Perl API for Apache 2.0 filtering framework.

Make sure to read the `Filtering tutorial|docs::2.0::user::handlers::filters`.

9.3 Common Filter API

The following methods can be called from any filter handler:

9.3.1 *c*

The current connection object can be retrieved from a connection or a request filter with:

```
$c = $f->c;
```

- **arg1: \$f (`Apache::Filter`)**
- **ret: \$c (`Apache::Connection`)**

9.3.2 *ctx*

Get and set the filter context data.

```
$ctx = $f->ctx;  
$f->ctx($ctx);
```

- **arg1: \$f (`Apache::Filter`)**
- **opt arg2: \$ctx (scalar)**

Could be any perl SCALAR.

- **ret: \$ctx (scalar)**

Could be any perl SCALAR.

A filter context is created before the filter is called for the first time and it's destroyed at the end of the request. The context is preserved between filter invocations of the same request. So if a filter needs to store some data between invocations it should use the filter context for that. The filter context is initialized with the `undef` value.

The `ctx` method accepts a single SCALAR argument. Therefore if you want to store any other perl data-structure you should use a reference to it.

For example you can store a hash reference:

```
$f->ctx({ foo => 'bar' });
```

and then access it:

```
$foo = $f->ctx->{foo};
```

if you access the context more than once it's more efficient to copy it's value before using it:

```
my $ctx = $f->ctx;
$foo = $ctx->{foo};
```

to avoid redundant method calls. As of this writing `$ctx` is not a tied variable, so if you modify it need to store it at the end:

```
$f->ctx($ctx);
```

META: later we might make it a TIED-variable interface, so it'll be stored automatically.

Besides its usage to store data between filter invocations, this method is also useful when as a flag. For example here is how to ensure that something happens only once during the filter's life:

```
unless ($f->ctx) {
    do_something_once();
    $f->ctx(1);
}
```

9.3.3 *freq*

Get/set the `Apache::FilterRec` (filter record) object.

```
my $freq = $f->freq();
$f->freq($freq);
```

- **arg1: \$f** (`Apache::Filter`)
- **opt arg2: \$freq** (`Apache::FilterRec`)
- **opt ret: \$freq** (`Apache::FilterRec`)

9.3.4 *next*

Returns the `Apache::Filter` object of the next filter in chain.

```
$next_f = $f->next;
```

- **arg1: \$f (Apache::Filter)**
- **ret: \$next_f (Apache::Filter)**

Since Apache inserts several core filters at the end of each chain, normally this method always returns an object. However if it's not a mod_perl filter handler, you can call only the following methods on it: `get_brigade`, `pass_brigade`, `c`, `r`, `frec` and `next`. If you call other methods the behavior is undefined.

META: I doubt anybody will ever need to mess with other filters, from within a mod_perl filter. but if the need arises it's easy to tell a mod_perl filter from non-mod_perl one by calling `$f->frec->name` (it'll return one of the following four names: *modperl_request_output*, *modperl_request_input*, *modperl_connection_output* or *modperl_connection_input*).

9.3.5 *r*

Inside an HTTP request filter retrieve the current request object:

```
$r = $f->r;
```

- **arg1: \$f (Apache::Filter)**
- **ret: \$r (Apache::RequestRec)**

If a sub-request adds filters, then the sub-request is the request associated with the filter.

9.3.6 *remove*

Remove the current filter from the filter chain (for the current request).

```
$f->remove;
```

- **arg1: \$f (Apache::Filter)**
- **ret: no return value**

Notice that you should either complete the current filter invocation normally (by calling `get_brigade` or `pass_brigade` depending on the filter kind) or if nothing was done, return `Apache::DECLINED` and mod_perl will take care of passing the current bucket brigade through unmodified to the next filter in chain.

note: calling `remove()` on the very top connection filter doesn't affect the filter chain due to a bug in Apache 2.0.46 and lower (may be fixed in 2.0.47). So don't use it with connection filters, till it gets fixed in Apache and then make sure to require the minimum Apache version if you rely on it.

9.4 Bucket Brigade Filter API

The following methods can be called from any filter, directly manipulating bucket brigades:

9.4.1 fflush

Flush the \$bb brigade down the filter stack.

```
$ret = $f->fflush($bb);
```

- **arg1: \$f (Apache::Filter)**

The current filter

- **arg2: \$bb (Apache::Filter)**

The brigade to flush

- **ret: XXX**

9.4.2 get_brigade

This is a method to use in bucket brigade input filters. It acquires a bucket brigade from the upstream input filter.

```
$ret = $next_f->get_brigade($bb, $mode, $block, $readbytes);
```

- **arg1: \$next_f (Apache::Filter)**

The next filter in the chain

- **arg2: \$bb (APR::Brigade)**

The original brigade passed to get_brigade() must be empty. On return it gets populated with the next bucket brigade, or nothing if there is no more data to read.

- **arg3: \$mode (integer)**

The way in which the data should be read

- **arg4: \$block (integer)**

How the operations should be performed APR::BLOCK_READ, APR::NONBLOCK_READ

- **arg5: \$readbytes (integer)**

How many bytes to read from the next filter.

- **ret: \$ret (integer)**

It returns APR::SUCCESS on success, otherwise a failure code, in which case it should be returned to the caller.

If the bottom-most filter doesn't read from the network, then `Apache::NOBODY_READ` is returned (META: need to add this constant).

For example:

```
sub filter {
    my($f, $bb, $mode, $block, $readbytes) = @_;

    my $rv = $f->next->get_brigade($bb, $mode, $block, $readbytes);
    return $rv unless $rv == APR::SUCCESS;

    # ... process $bb

    return Apache::OK;
}
```

Normally arguments `$mode`, `$block`, `$readbytes` are the same as passed to the filter itself.

It returns `APR::SUCCESS` on success, otherwise a failure code, in which case it should be returned to the caller.

9.4.3 *pass_brigade*

This is a method to use in bucket brigade output filters. It passes the current bucket brigade to the downstream output filter.

```
$ret = $next_f->pass_brigade($bb);
```

- **arg1: \$next_f (Apache::Filter)**

The next filter in the chain

- **arg2: \$bb (APR::Brigade)**

The current bucket brigade

- **ret: \$ret (integer)**

It returns `APR::SUCCESS` on success, otherwise a failure code, in which case it should be returned to the caller.

If the bottom-most filter doesn't write to the network, then `Apache::NOBODY_WROTE` is returned (META: need to add this constant).

The caller relinquishes ownership of the brigade (i.e. it may get destroyed/overwritten/etc by the callee).

For example:

```

sub filter {
    my($f, $bb) = @_;

    # ... process $bb

    my $rv = $f->next->pass_brigade($bb);
    return $rv unless $rv == APR::SUCCESS;

    # process $bb
    return Apache::OK;
}

```

9.5 Streaming Filter API

The following methods can be called from any filter, which uses the simplified streaming functionality:

9.5.1 *seen_eos*

This methods returns a true value when the EOS bucket is seen by the `read` method.

```
$ret = $f->seen_eos;
```

- **arg1: \$f (Apache::Filter)**

The filter to remove

- **ret: \$ret (integer)**

a true value if seen, otherwise a false value

This method only works in streaming filters which exhaustively `$f->read` all the incoming data in a while loop, like so:

```

while ($f->read(my $buffer, $read_len)) {
    # do something with $buffer
}
if ($f->seen_eos) {
    # do something
}

```

This method is useful when a streaming filter wants to append something to the very end of data, or do something at the end of the last filter invocation. After the EOS bucket is read, the filter should expect not to be invoked again.

If an input streaming filter doesn't consume all data in the bucket brigade (or even in several bucket brigades), it has to generate the EOS event by itself. So when the filter is done it has to set the EOS flag:

```
$f->seen_eos(1);
```

when the filter handler returns, internally `mod_perl` will take care of creating and sending the EOS bucket to the upstream input filter.

A similar logic may apply for output filters.

In most other cases you shouldn't set this flag. When this flag is prematurely set (before the real EOS bucket has arrived) in the current filter invocation, instead of invoking the filter again, `mod_perl` will create and send the EOS bucket to the next filter, ignoring any other bucket brigades that may have left to consume. As mentioned earlier this special behavior is useful in writing special tests that test abnormal situations.

9.5.2 *read*

Read data from the filter

```
$ret = $f->read(my $buffer, $read_len);
```

- **arg1: \$f (Apache::Filter)**
- **arg2: \$buffer (scalar)**
- **arg3: \$read_len (integer)**
- **ret: \$ret (number)**

Reads at most `$read_len` characters into `$buffer`. It returns a true value as long as it had something to read, or a false value otherwise.

This is a streaming filter method, which acquires a single bucket brigade behind the scenes and reads data from all its buckets. Therefore it can only read from one bucket brigade per filter invocation.

If the EOS bucket is read, the `seen_eos` method will return a true value.

9.5.3 *fputs*

META: Autogenerated - needs to be reviewed/completed

```
$ret = $f->fputs($bb, $str);
```

- **arg1: \$f (Apache::Filter)**
- **arg2: \$bb (APR::Brigade)**
- **arg3: \$str (string)**
- **ret: \$ret (integer)**

9.5.4 *print*

Send the contents of `$buffer` to the next filter in chain (via internal buffer).

```
$f->print($buffer);
```

- **arg1:** `$f` (**Apache::Filter**)
- **arg2:** `$buffer` (scalar)
- **ret:** XXX

This method should be used only in streaming filters.

9.6 Other Filter-related API

Other methods which affect filters, but called on non-`Apache::Filter` objects:

9.6.1 *add_input_filter*

Add `&callback` filter handler to input request filter chain.

```
$r->add_input_filter(\&callback);
```

Add `&callback` filter handler to input connection filter chain.

```
$c->add_input_filter(\&callback);
```

- **arg1:** `$c` (**Apache::Connection**) or `$r` (**Apache::RequestRec**)
- **arg2:** `&callback` (CODE ref)
- **ret:** XXX

9.6.2 *add_output_filter*

Add `&callback` filter handler to output request filter chain.

```
$r->add_output_filter(\&callback);
```

Add `&callback` filter handler to output connection filter chain.

```
$c->add_output_filter(\&callback);
```

- **arg1:** `$c` (**Apache::Connection**) or `$r` (**Apache::RequestRec**)
- **arg2:** `&callback` (CODE ref)
- **ret:** XXX

9.7 TIE Interface

`Apache::Filter` also implements a tied interface, so you can work with the `$f` object as a hash reference.

META: complete

9.7.1 *TIEHANDLE*

META: Autogenerated - needs to be reviewed/completed

```
$ret = TIEHANDLE($stashsv, $sv);
```

- **arg1:** `$stashsv` (scalar)
- **arg2:** `$sv` (scalar)
- **ret:** `$ret` (scalar)

9.7.2 *PRINT*

META: Autogenerated - needs to be reviewed/completed

```
$ret = PRINT(...);
```

- **arg1:** `...` (XXX)
- **ret:** `$ret` (integer)

9.8 Filter Handler Attributes

Packages using filter attributes have to subclass `Apache::Filter`:

```
package MyApache::FilterCool;
use base qw(Apache::Filter);
```

Attributes are parsed during the code compilation, by the function `MODIFY_CODE_ATTRIBUTES`, inherited from the `Apache::Filter` package.

9.8.1 *FilterRequestHandler*

The `FilterRequestHandler` attribute tells `mod_perl` to insert the filter into an HTTP request filter chain.

For example, to configure an output request filter handler, use the `FilterRequestHandler` attribute in the handler subroutine's declaration:

```
package MyApache::FilterOutputReq;
sub handler : FilterRequestHandler { ... }
```

and add the configuration entry:

```
PerlOutputFilterHandler MyApache::FilterOutputReq
```

This is the default mode. So if you are writing an HTTP request filter, you don't have to specify this attribute.

The section HTTP Request vs. Connection Filters delves into more details.

9.8.2 *FilterConnectionHandler*

The `FilterConnectionHandler` attribute tells `mod_perl` to insert this filter into a connection filter chain.

For example, to configure an output connection filter handler, use the `FilterConnectionHandler` attribute in the handler subroutine's declaration:

```
package MyApache::FilterOutputCon;
sub handler : FilterConnectionHandler { ... }
```

and add the configuration entry:

```
PerlOutputFilterHandler MyApache::FilterOutputCon
```

The section HTTP Request vs. Connection Filters delves into more details.

9.8.3 *FilterInitHandler*

The attribute `FilterInitHandler` marks the function suitable to be used as a filter initialization callback, which is called immediately after a filter is inserted to the filter chain and before it's actually called.

```
sub init : FilterInitHandler {
    my $f = shift;
    #...
    return Apache::OK;
}
```

In order to hook this filter callback, the real filter has to assign this callback using the `FilterHasInitHandler` which accepts a reference to the callback function.

For further discussion and examples refer to the Filter Initialization Phase tutorial section.

9.8.4 *FilterHasInitHandler*

If a filter wants to run an initialization callback it can register such using the `FilterHasInitHandler` attribute. Similar to `push_handlers` the callback reference is expected, rather than a callback name. The used callback function has to have the `FilterInitHandler` attribute. For example:

```
package MyApache::FilterBar;
use base qw(Apache::Filter);
sub init    : FilterInitHandler { ... }
sub filter : FilterRequestHandler FilterHasInitHandler(&init) {
    my ($f, $bb) = @_;
    # ...
    return Apache::OK;
}
```

For further discussion and examples refer to the Filter Initialization Phase tutorial section.

9.9 Configuration

mod_perl 2.0 filters configuration is explained in the filter handlers tutorial.

9.9.1 PerlInputFilterHandler

See PerlInputFilterHandler.

9.9.2 PerlOutputFilterHandler

See PerlOutputFilterHandler.

9.9.3 PerlSetInputFilter

See PerlSetInputFilter.

9.9.4 PerlSetOutputFilter

See PerlSetInputFilter.

9.10 See Also

mod_perl 2.0 documentation.

9.11 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

9.12 Authors

The mod_perl development team and numerous contributors.

10 Apache::FilterRec - Perl API for manipulating the Apache filter record

10.1 Synopsis

```
use Apache::FilterRec ();
```

META: to be completed

10.2 Description

META: to be completed

10.3 API

`Apache::FilterRec` provides the following functions and/or methods:

10.3.1 name

META: Autogenerated - needs to be reviewed/completed

The registered name for this filter

```
$name = $frec->name();
```

- **arg1:** `$frec` (`Apache::FilterRec`)
- **ret:** `$name` (string)

10.3.2 next

META: Autogenerated - needs to be reviewed/completed

The next `filter_rec` in the list

```
$next_frec = $frec->next();
```

- **arg1:** `$frec` (`Apache::FilterRec`)
- **ret:** `$next_frec` (`Apache::FilterRec`)

10.4 See Also

`mod_perl 2.0` documentation.

10.5 Copyright

`mod_perl 2.0` and its core modules are copyrighted under The Apache Software License, Version 1.1.

10.6 Authors

The mod_perl development team and numerous contributors.

11 Apache::HookRun - Perl API for XXX

11.1 Synopsis

```
use Apache::HookRun ( );
```

META: to be completed

11.2 Description

META: to be completed

11.3 API

Apache::HookRun provides the following functions and/or methods:

11.3.1 die

META: Autogenerated - needs to be reviewed/completed

Kill the current request

```
$r->die($type);
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **arg2: \$type (XXX)**

Why the request is dieing

- **ret: no return value**

11.3.2 invoke_handler

META: Autogenerated - needs to be reviewed/completed

Run the handler phase of each module until a module accepts the responsibility of serving the request

```
$ret = $r->invoke_handler();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

The status of the current request

11.3.3 run_access_checker

META: Autogenerated - needs to be reviewed/completed

This hook is used to apply additional access control to this resource. It runs **before** a user is authenticated, so this hook is really to apply additional restrictions independent of a user. It also runs independent of 'Require' directive usage.

```
$ret = $r->run_access_checker();
```

- **arg1: \$r (Apache::RequestRec)**

the current request

- **ret: \$ret (integer)**

Apache::OK, Apache::DECLINED, or Apache::HTTP_...

11.3.4 run_auth_checker

META: Autogenerated - needs to be reviewed/completed

This hook is used to check to see if the resource being requested is available for the authenticated user (r->user and r->ap_auth_type). It runs after the access_checker and check_user_id hooks. Note that it will **only** be called if Apache determines that access control has been applied to this resource (through a 'Require' directive).

```
$ret = $r->run_auth_checker();
```

- **arg1: \$r (Apache::RequestRec)**

the current request

- **ret: \$ret (integer)**

Apache::OK, Apache::DECLINED, or Apache::HTTP_...

11.3.5 run_check_user_id

META: Autogenerated - needs to be reviewed/completed

This hook is used to analyze the request headers, authenticate the user, and set the user information in the request record (r->user and r->ap_auth_type). This hook is only run when Apache determines that authentication/authorization is required for this resource (as determined by the 'Require' directive). It runs after the access_checker hook, and before the auth_checker hook.

```
$ret = $r->run_check_user_id();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

Apache::OK, Apache::DECLINED, or Apache::HTTP_...

11.3.6 run_create_request

META: Autogenerated - needs to be reviewed/completed

Gives modules a chance to create their request_config entry when the request is created.

```
$ret = $r->run_create_request();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

Apache::OK, Apache::DECLINED, or Apache::HTTP_...

11.3.7 run_fixups

META: Autogenerated - needs to be reviewed/completed

Allows modules to perform module-specific fixing of header fields. This is invoked just before any content-handler

```
$ret = $r->run_fixups();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

Apache::OK, Apache::DECLINED, or Apache::HTTP_...

11.3.8 run_handler

META: Autogenerated - needs to be reviewed/completed

Run the handler functions for each module

```
$ret = $r->run_handler();
```

- **arg1: \$r (Apache::RequestRec)**

The request_rec

- **ret: \$ret (integer)**

non-wildcard handlers should HOOK_MIDDLE, wildcard HOOK_LAST

11.3.9 run_header_parser

META: Autogenerated - needs to be reviewed/completed

Run the header parser functions for each module

```
$ret = $r->run_header_parser();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

Apache::OK or Apache::DECLINED

11.3.10 run_log_transaction

META: Autogenerated - needs to be reviewed/completed

This hook allows modules to perform any module-specific logging activities over and above the normal server things.

```
$ret = $r->run_log_transaction();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

Apache::OK, Apache::DECLINED, or Apache::HTTP_...

11.3.11 run_map_to_storage

META: Autogenerated - needs to be reviewed/completed

This hook allow modules to set the per_dir_config based on their own context (such as <Proxy> sections) and responds to contextless requests such as TRACE that need no security or filesystem mapping. based on the filesystem.

```
$ret = $r->run_map_to_storage();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

Apache::DONE (or Apache::HTTP_) if this contextless request was just fulfilled (such as TRACE), OK if this is not a file, and Apache::DECLINED if this is a file. The core map_to_storage (HOOK_RUN_LAST) will directory_walk and file_walk the r->filename.

11.3.12 run_post_read_request

META: Autogenerated - needs to be reviewed/completed

post_read_request --- run right after read_request or internal_redirect, and not run during any subrequests. This hook allows modules to affect the request immediately after the request has been read, and before any other phases have been processes. This allows modules to make decisions based upon the input header fields

```
$ret = $r->run_post_read_request();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

Apache::OK or Apache::DECLINED

11.3.13 run_translate_name

META: Autogenerated - needs to be reviewed/completed

This hook allow modules an opportunity to translate the URI into an actual filename. If no modules do anything special, the server's default rules will be followed.

11.4 See Also

```
$ret = $r->run_translate_name();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

Apache::OK, Apache::DECLINED, or Apache::HTTP_...

11.3.14 run_type_checker

META: Autogenerated - needs to be reviewed/completed

This routine is called to determine and/or set the various document type information bits, like Content-type (via `r->content_type`), language, etc.

```
$ret = $r->run_type_checker();
```

- **arg1: \$r (Apache::RequestRec)**

the current request

- **ret: \$ret (integer)**

Apache::OK, Apache::DECLINED, or Apache::HTTP_...

11.4 See Also

`mod_perl 2.0 documentation`.

11.5 Copyright

`mod_perl 2.0` and its core modules are copyrighted under The Apache Software License, Version 1.1.

11.6 Authors

The `mod_perl` development team and numerous contributors.

12 Apache::Log - Perl API for Apache Logging Methods

12.1 Synopsis

```
#in startup.pl
#-----
use Apache::Log;

use Apache::Const -compile => qw(OK :log);
use APR::Const    -compile => qw(:error SUCCESS);

my $s = Apache->server;

$s->log_error("server: log_error");
$s->log_error(__FILE__, __LINE__, Apache::LOG_ERR,
              APR::SUCCESS, "log_error logging at err level");
$s->log_error(Apache::Log::LOG_MARK, Apache::LOG_DEBUG,
              APR::ENOTIME, "debug print");
Apache::Server->log_error("routine warning");

Apache->warn("routine warning");
Apache::warn("routine warning");
Apache::Server->warn("routine warning");

#in a handler
#-----
use Apache::Log;

use strict;
use warnings FATAL => 'all';

use Apache::Const -compile => qw(OK :log);
use APR::Const    -compile => qw(:error SUCCESS);

sub handler{
    my $r = shift;
    $r->log_error("request: log_error");
    $r->warn("whoah!");

    my $rlog = $r->log;
    for my $level qw(emerg alert crit error warn notice info debug) {
        no strict 'refs';
        $rlog->$level($package, "request: $level log level");
    }

    # can use server methods as well
    my $s = $r->server;
    $s->log_error("server: log_error");

    $r->log_rerror(Apache::Log::LOG_MARK, Apache::LOG_DEBUG,
                  APR::ENOTIME, "in debug");

    $s->log_serror(Apache::Log::LOG_MARK, Apache::LOG_INFO,
                  APR::SUCESS, "server info");

    $s->log_serror(Apache::Log::LOG_MARK, Apache::LOG_ERR,
                  APR::ENOTIME, "fatal error");
}
```

```
$s->warn('routine server warning');  
  
    return Apache::OK;  
}
```

12.2 Description

Apache::Log provides the Perl API for Apache logging methods.

Depending on the the current `LogLevel` setting, only logging with the same log level or higher will be loaded. For example if the current `LogLevel` is set to *warning*, only messages with log level of the level *warning* or higher (*err*, *crit*, *elert* and *emerg*) will be logged. Therefore this:

```
$r->log_error(Apache::Log::LOG_MARK, Apache::LOG_WARNING,  
             APR::ENOTIME, "warning!");
```

will log the message, but this one won't:

```
$r->log_error(Apache::Log::LOG_MARK, Apache::LOG_INFO,  
             APR::ENOTIME, "just an info");
```

It will be logged only if the server log level is set to *info* or *debug*. `LogLevel` is set in the configuration file, but can be changed using the `$s->loglevel()` method.

The filename and the line number of the caller are logged only if `Apache::LOG_DEBUG` is used (because that's how Apache 2.0 logging mechanism works).

12.3 Constants

Log level constants can be compiled all at once:

```
use Apache::Const -compile => qw(:log);
```

or individually:

```
use Apache::Const -compile => qw(LOG_DEBUG LOG_INFO);
```

12.3.1 *LogLevel Constants*

The following constants (sorted from the most severe level to the least severe) are used in logging methods to specify the log level at which the message should be logged:

12.3.1.1 **Apache::LOG_EMERG**

12.3.1.2 Apache::LOG_ALERT**12.3.1.3 Apache::LOG_CRIT****12.3.1.4 Apache::LOG_ERR****12.3.1.5 Apache::LOG_WARNING****12.3.1.6 Apache::LOG_NOTICE****12.3.1.7 Apache::LOG_INFO****12.3.1.8 Apache::LOG_DEBUG**

Make sure to compile the APR status constants before using them. For example to compile `APR::SUCCESS` and all the APR error status constants do:

```
use APR::Const    -compile => qw(:error SUCCESS);
```

12.3.2 Other Constants**12.3.2.1 Apache::LOG_LEVELMASK**

used to mask off the level value, to make sure that the log level's value is within the proper bits range. e.g.:

```
$loglevel &= LOG_LEVELMASK;
```

12.3.2.2 Apache::LOG_TOCLIENT

used to give content handlers the option of including the error text in the `ErrorDocument` sent back to the client. When `Apache::LOG_TOCLIENT` is passed to `log_error()` the error message will be saved in the `$r`'s notes table, keyed to the string *"error-notes"*, if and only if the severity level of the message is `Apache::LOG_WARNING` or greater and there are no other *"error-notes"* entry already set in the request record's notes table. Once the *"error-notes"* entry is set, it is up to the error handler to determine whether this text should be sent back to the client. For example:

```
$r->log_error(Apache::Log::LOG_MARK, Apache::LOG_ERR|Apache::LOG_TOCLIENT,
              APR::ENOTIME, "request log_error");
```

now the log message can be retrieved via:

```
$r->notes->get("error-notes");
```

Remember that client-generated text streams sent back to the client **MUST** be escaped to prevent CSS attacks.

12.3.2.3 Apache::LOG_STARTUP

is useful for startup message where no timestamps, logging level is wanted. For example:

```
$s->log_error(Apache::Log::LOG_MARK, Apache::LOG_INFO,
              APR::SUCCESS, "This log message comes with a header");
```

will print:

```
[Wed May 14 16:47:09 2003] [info] This log message comes with a header
```

whereas, when Apache::LOG_STARTUP is binary ORed as in:

```
$s->log_error(Apache::Log::LOG_MARK, Apache::LOG_INFO|Apache::LOG_STARTUP,
              APR::SUCCESS, "This log message comes with no header");
```

then the logging will be:

```
This log message comes with no header
```

12.4 Server Logging Methods

12.4.1 *\$s->log_error*

just logs the supplied message to *error_log*

```
$s->log_error(@message);
```

- **arg1: \$s (Apache::Server)**
- **arg2: @message (ARRAY)**

what to log

- **ret: (XXX)**

For example:

```
$s->log_error("running low on memory");
```

12.4.2 *\$s->log_serror*

This function provides a fine control of when the message is logged, gives an access to built-in status codes.

```
$s->log_serror($file, $line, $level, $status, @message);
```

- **arg1: \$s (Apache::Server)**
- **arg2: \$file (string)**

The file in which this function is called

- **arg3: \$line (number)**

The line number on which this function is called

- **arg4: \$level (Apache::LOG_* constant)**

The level of this error message

- **arg5: \$status (integer)**

The status code from the last command (similar to \$! in perl, usually APR:: status constant)

- **arg6: @message (ARRAY)**

The log message

- **ret: (XXX)**

For example:

```
$s->log_error(Apache::Log::LOG_MARK, Apache::LOG_ERR,
              APR::SUCCESS, "log_error logging at err level");

$s->log_error(Apache::Log::LOG_MARK, Apache::LOG_DEBUG,
              APR::ENOTIME, "debug print");
```

12.4.3 \$s->log

returns a log handle which can be used to log messages of different levels.

```
my $slog = $s->log;
```

- **arg1: \$s (Apache::Server)**
- **ret: \$slog (scalar)**

12.5 Request Logging Methods

12.5.1 \$r->log_error

logs the supplied message (similar to \$s->log_error).

```
$r->log_error(@message);
```

- **arg1: \$r (Apache::RequestRec)**
- **arg2: @message (ARRAY)**

what to log

- **ret: (XXX)**

For example:

```
$r->log_error("the request is about to end");
```

12.5.2 *\$r->log_error*

This function provides a fine control of when the message is logged, gives an access to built-in status codes.

```
$r->log_error($file, $line, $level, $status, @message);
```

arguments are identical to `$s->log_serror`.

For example:

```
$r->log_error(Apache::Log::LOG_MARK, Apache::LOG_ERR,
             APR::SUCCESS, "log_error logging at err level");

$r->log_error(Apache::Log::LOG_MARK, Apache::LOG_DEBUG,
             APR::ENOTIME, "debug print");
```

12.5.3 *\$r->log*

returns a log handle which can be used to log messages of different levels.

```
$rlog = $r->log;
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$rlog (scalar)**

12.6 Other Logging Methods

12.6.1 *LogLevel Methods*

after getting the log handle with `$s->log` or `$r->log`, use one of the following methods (corresponding to the LogLevel levels):

```
emerg(), alert(), crit(), error(), warn(), notice(), info(), debug()
```

to control when messages should be logged:

```
$s->log->emerg(@message);
$r->log->emerg(@message);
```

- **arg1: \$slog (log handle)**
- **arg2: @message (ARRAY)**

For example if the LogLevel is error and the following code is executed:

```
my $slog = $s->log;  
$slog->debug("just ", "some debug info");  
$slog->warn(@warnings);  
$slog->crit("dying");
```

only the last command's logging will be performed. This is because *warn*, *debug* and other logging command which are listed right to *error* will be disabled.

12.6.2 emerg

See LogLevel Methods.

12.6.3 alert

See LogLevel Methods.

12.6.4 crit

See LogLevel Methods.

12.6.5 error

See LogLevel Methods.

12.6.6 warn

See LogLevel Methods.

12.6.7 notice

See LogLevel Methods.

12.6.8 info

See LogLevel Methods.

12.6.9 *debug*

See LogLevel Methods.

12.7 General Functions

12.7.1 *Apache::Log::LOG_MARK*

```
( $file, $line ) = Apache::Log::LOG_MARK();
```

Though looking like a constant, this is a function, which returns a list of two items: (`__FILE__`, `__LINE__`), i.e. the file and the line where the function was called from.

It's mostly useful to be passed as the first argument to those logging methods, expecting the filename and the line number as the first arguments (e.g., `$s->log_serror` and `$r->log_rerror`).

12.7.2 *Apache::Log::log_pid*

Log the current pid of the parent process

```
Apache::Log::log_pid($pool, $fname);
```

- **arg1: \$p (APR::Pool)**

The pool to use for logging

- **arg2: \$fname (APR::Pool)**

The name of the file to log to

- **ret: no return value**

12.8 Aliases

12.8.1 *\$s->warn*

```
$s->warn(@warnings);
```

is the same as:

```
$s->log_error(Apache::Log::LOG_MARK, Apache::LOG_WARNING,
             APR::SUCCESS, @warnings)
```

For example:

12.9 See Also

```
$s->warn('routine server warning');
```

12.8.2 Apache->warn

12.8.3 Apache::warn

```
Apache->warn(@warnings);
```

12.9 See Also

mod_perl 2.0 documentation.

12.10 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

12.11 Authors

The mod_perl development team and numerous contributors.

13 Apache::Module - Perl API for creating and working with Apache modules

13.1 Synopsis

```
use Apache::Module ();
```

META: to be completed

13.2 Description

META: to be completed

See Apache Server Configuration Customization in Perl.

13.3 API

`Apache::Module` provides the following functions and/or methods:

13.3.1 find_linked_module

META: Autogenerated - needs to be reviewed/completed

Find a module based on the name of the module

```
$ret = find_linked_module($name);
```

- **arg1: \$name (string)**

the name of the module

- **ret: \$ret (Apache::Module)**

the module structure if found, NULL otherwise

13.3.2 find_module_name

META: Autogenerated - needs to be reviewed/completed

Find the name of the specified module

```
$ret = $module->find_module_name();
```

- **arg1: \$module (Apache::Module)**

The module to get the name for

- **ret: \$ret (string)**

the name of the module

13.3.3 remove_loaded_module

META: Autogenerated - needs to be reviewed/completed

Remove a module from the chained modules list and the list of loaded modules

```
$module->remove_loaded_module();
```

- **arg1: \$module (Apache::Module)**
- **ret: no return value**

13.3.4 remove_module

META: Autogenerated - needs to be reviewed/completed

Remove a module from the server. There are some caveats: when the module is removed, its slot is lost so all the current per-dir and per-server configurations are invalid. So we should only ever call this function when you are invalidating almost all our current data. I.e. when doing a restart.

```
$module->remove_module();
```

- **arg1: \$module (Apache::Module)**
the module structure of the module to remove
- **ret: no return value**

13.3.5 top_module

META: Autogenerated - needs to be reviewed/completed

```
$ret = Apache::Module->top_module();
```

- **arg1: Apache::Module (class name)**
- **ret: \$ret (Apache::Module)**

13.3.6 version

META: Autogenerated - needs to be reviewed/completed

API version, **not** module version; check that module is compatible with this version of the server.

```
$version = $module->version();
```

- **arg1:** `$module (Apache::Module)`
- **ret:** `$version (integer)`

13.3.7 minor_version

META: Autogenerated - needs to be reviewed/completed

API minor version. Provides API feature milestones. Not checked during module init

```
$minor_version = $module->minor_version();
```

- **arg1:** `$module (Apache::Module)`
- **ret:** `$minor_version (integer)`

13.3.8 module_index

META: Autogenerated - needs to be reviewed/completed

Index to this modules structures in config vectors.

```
$index = $module->module_index();
```

- **arg1:** `$module (Apache::Module)`
- **ret:** `$index (integer)`

13.3.9 name

META: Autogenerated - needs to be reviewed/completed

The name of the module's C file

```
$name = $module->name();
```

- **arg1:** `$module (Apache::Module)`
- **ret:** `$name (string)`

13.3.10 dynamic_load_handle

META: Autogenerated - needs to be reviewed/completed

The handle for the DSO. Internal use only

```
$dl_handle = $module->dynamic_load_handle();
```

- **arg1:** `$module (Apache::Module)`
- **ret:** `$dl_handle (SCALAR)`

13.3.11 next

META: Autogenerated - needs to be reviewed/completed

A pointer to the next module in the list

```
$next_module = $module->next();
```

- **arg1:** `$module (Apache::Module)`
- **ret:** `$next_module (Apache::Module)`

13.3.12 cmds

META: Autogenerated - needs to be reviewed/completed

A `command_rec` table that describes all of the directives this module defines.

```
$cmd_rec = $module->cmds();
```

- **arg1:** `$module (Apache::Module)`
- **ret:** `$cmd_rec (Apache::Command)`

13.4 See Also

`mod_perl 2.0` documentation.

13.5 Copyright

`mod_perl 2.0` and its core modules are copyrighted under The Apache Software License, Version 1.1.

13.6 Authors

The `mod_perl` development team and numerous contributors.

14 Apache::PerlSections - Default Handler for Perl sections

14.1 Synopsis

```
<Perl >
@PerlModule = qw(Mail::Send Devel::Peek);

#run the server as whoever starts it
$User = getpwuid(>) || >;
$Group = getgrgid()) || );

$ServerAdmin = $User;

</Perl>
```

14.2 Description

With `<Perl >...</Perl>` sections, it is possible to configure your server entirely in Perl.

`<Perl >` sections can contain *any* and as much Perl code as you wish. These sections are compiled into a special package whose symbol table `mod_perl` can then walk and grind the names and values of Perl variables/structures through the Apache core configuration gears.

Block sections such as `<Location>..</Location>` are represented in a `%Location` hash, e.g.:

```
<Perl>
$Location{"/~doug/"} = {
    AuthUserFile => '/tmp/htpasswd',
    AuthType     => 'Basic',
    AuthName     => 'test',
    DirectoryIndex => [qw(index.html index.htm)],
    Limit        => {
        METHODS => 'GET POST',
        require => 'user dougm',
    },
};
</Perl>
```

If an Apache directive can take two or three arguments you may push strings (the lowest number of arguments will be shifted off the `@list`) or use an array reference to handle any number greater than the minimum for that directive:

```
push @Redirect, "/foo", "http://www.foo.com/";

push @Redirect, "/imdb", "http://www.imdb.com/";

push @Redirect, [qw(temp "/here" "http://www.there.com")];
```

Other section counterparts include `%VirtualHost`, `%Directory` and `%Files`.

To pass all environment variables to the children with a single configuration directive, rather than listing each one via `PassEnv` or `PerlPassEnv`, a `<Perl >` section could read in a file and:

```
push @PerlPassEnv, [$key => $val];
```

or

```
Apache->httpd_conf("PerlPassEnv $key $val");
```

These are somewhat simple examples, but they should give you the basic idea. You can mix in any Perl code you desire. See *eg/httpd.conf.pl* and *eg/perl_sections.txt* in the `mod_perl` distribution for more examples.

Assume that you have a cluster of machines with similar configurations and only small distinctions between them: ideally you would want to maintain a single configuration file, but because the configurations aren't *exactly* the same (e.g. the `ServerName` directive) it's not quite that simple.

`<Perl >` sections come to rescue. Now you have a single configuration file and the full power of Perl to tweak the local configuration. For example to solve the problem of the `ServerName` directive you might have this `<Perl >` section:

```
<Perl >
$ServerName = 'hostname';
</Perl>
```

For example if you want to allow personal directories on all machines except the ones whose names start with *secure*:

```
<Perl >
$ServerName = 'hostname';
if ($ServerName !~ /^secure/) {
    $UserDir = "public.html";
}
else {
    $UserDir = "DISABLED";
}
</Perl>
```

14.3 Configuration Variables

There are a few variables that can be set to change the default behaviour of `<Perl >` sections.

14.3.1 *`$Apache::Server::SaveConfig`*

By default, the namespace in which `<Perl >` sections are evaluated is cleared after each block closes. By setting it to a true value, the content of those namespaces will be preserved and will be available for inspection by modules like `Apache::Status`.

14.3.2 *\$Apache::Server::StrictPerlSections*

By default, compilation and run-time errors within `<Perl >` sections will cause a warning to be printed in the `error_log`. By setting this variable to a true value, code in the sections will be evaluated as if "use strict" was in usage, and all warning and errors will cause the server to abort startup and report the first error.

14.4 Advanced API

`mod_perl 2.0` now introduces the same general concept of handlers to `<Perl >` sections. `Apache::PerlSections` simply being the default handler for them.

To specify a different handler for a given perl section, an extra handler argument must be given to the section:

```
<Perl handler="My::PerlSection::Handler" somearg="test1">
    $foo = 1;
    $bar = 2;
</Perl>
```

And in `My/PerlSection/Handler.pm`:

```
sub My::Handler::handler : handler {
    my($self, $parms, $args) = @_;
    #do your thing!
}
```

So, when that given `<Perl >` block is encountered, the code within will first be evaluated, then the handler routine will be invoked with 3 arguments

`$self` is self-explanatory

`$parms` is the `Apache::CmdParms` for this Container, for example, you might want to call `$parms->server()` to get the current server.

`$args` is an `APR::Table` object of the section arguments, the 2 guaranteed ones will be:

```
$args->{'handler'} = 'My::PerlSection::Handler';
$args->{'package'} = 'Apache::ReadConfig';
```

Other `name="value"` pairs given on the `<Perl >` line will also be included.

At this point, it's up to the handler routing to inspect the namespace of the `$args->{'package'}` and chooses what to do.

The most likely thing to do is to feed configuration data back into apache. To do that, use `Apache::Server->add_config("directive")`, for example:

```
$parms->server->add_config("Alias /foo /bar");
```

Would create a new alias. The source code of `Apache::PerlSections` is a good place to look for a practical example.

14.5 Bugs

14.5.1 *<Perl> directive missing closing '>'*

httpd-2.0.47 and earlier had a bug in the configuration parser which caused the startup failure with the following error:

```
Starting httpd:
Syntax error on line ... of /etc/httpd/conf/httpd.conf:
<Perl> directive missing closing '>'      [FAILED]
```

This has been fixed in httpd-2.0.48. If you can't upgrade to this or a higher version, please add a space before the closing '>' of the opening tag as a workaround. So if you had:

```
<Perl>
# some code
</Perl>
```

change it to be:

```
<Perl >
# some code
</Perl>
```

14.6 See Also

`mod_perl 2.0` documentation.

14.7 Copyright

`mod_perl 2.0` and its core modules are copyrighted under The Apache Software License, Version 1.1.

14.8 Authors

The `mod_perl` development team and numerous contributors.

15 Apache::Process - Perl API for XXX

15.1 Synopsis

```
use Apache::Process ();
```

META: to be completed

15.2 Description

META: to be completed

15.3 API

`Apache::Process` provides the following functions and/or methods:

15.3.1 *pool*

META: Autogenerated - needs to be reviewed/completed

Global pool. Cleared upon normal exit

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

- **arg1:** `$proc (Apache::Process)`
- **ret:** `$p (APR::Pool)`

15.3.2 *pconf*

META: Autogenerated - needs to be reviewed/completed

Configuration pool. Cleared upon restart

```
$p = $proc->pconf($newval);
```

- **arg1:** `$proc (Apache::Process)`
- **ret:** `$p (APR::Pool)`

15.3.3 *short_name*

META: Autogenerated - needs to be reviewed/completed

The name of the program used to execute the program

```
$short_name = $proc->short_name();
```


- **arg1: \$proc** (**Apache::Process**)
- **ret: \$short_name** (**string**)

15.4 See Also

mod_perl 2.0 documentation.

15.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

15.6 Authors

The mod_perl development team and numerous contributors.

16 Apache::RequestIO - Perl API for Apache request record IO

16.1 Synopsis

```
use Apache::RequestIO ();
```

META: to be completed

16.2 Description

Apache::RequestIO provides the API to perform IO on the Apache request object.

16.3 API

Apache::RequestIO provides the following functions and/or methods:

16.3.1 *discard_request_body*

META: Autogenerated - needs to be reviewed/completed

In HTTP/1.1, any method can have a body. However, most GET handlers wouldn't know what to do with a request body if they received one. This helper routine tests for and reads any message body in the request, simply discarding whatever it receives. We need to do this because failing to read the request body would cause it to be interpreted as the next request on a persistent connection.

```
$ret = $r->discard_request_body();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

error status if request is malformed, Apache::OK otherwise

16.3.2 *setup_client_block*

META: Autogenerated - needs to be reviewed/completed

META: I think this method is deprecated along with other client_block methods, use plain \$r-<read() instead.

Setup the client to allow Apache to read the request body.

```
$ret = $r->setup_client_block($read_policy);
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **arg2: \$read_policy (Apache::RequestRec)**

How the server should interpret a chunked transfer-encoding. One of:

REQUEST_NO_BODY	Send 413 error if message has any body
REQUEST_CHUNKED_ERROR	Send 411 error if body without Content-Length
REQUEST_CHUNKED_DECHUNK	If chunked, remove the chunks for me.

- **ret: \$ret (integer)**

either OK or an error code

16.3.3 *should_client_block*

META: Autogenerated - needs to be reviewed/completed

META: I think this method is deprecated along with other client_block methods, use plain \$r-<read() instead.

Determine if the client has sent any data. This also sends a 100 Continue response to HTTP/1.1 clients, so modules should not be called until the module is ready to read content.

```
$ret = $r->should_client_block();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

0 if there is no message to read, 1 otherwise

16.3.4 *print*

Send data to the client.

```
$ret = $r->print(@msg);
```

- **arg1: \$r (Apache::RequestRec)**
- **arg2: @msg (ARRAY)**
- **ret: \$ret (number)**

16.3.5 *read*

Read data from the client.

```
$read_count = $r->read($buffer, $len, $offset);
```

META: same as CORE::read, minus the filehandle argument

- **arg1: \$r (Apache::RequestRec)**
- **arg2: \$buffer (scalar)**
- **arg3: \$len (scalar)**
- **arg4: \$offset (number)**
- **ret: \$read_count (number)**

How many characters were actually read

16.3.6 *rflush*

Flush any buffered data to the client.

```
$ret = $r->rflush();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$ret (integer)**

Unless `$| > 0`, data sent via `$r->print()` is buffered. This method flushes that data to the client.

16.3.7 *sendfile*

META: Autogenerated - needs to be reviewed/completed

```
$ret = $r->sendfile($filename, $offset, $len);
```

- **arg1: \$r (Apache::RequestRec)**
- **arg2: \$filename (Apache::RequestRec)**
- **arg3: \$offset (string)**
- **arg4: \$len (integer)**
- **ret: \$ret (integer)**

16.3.8 *write*

META: Autogenerated - needs to be reviewed/completed

Write data to the client

```
$ret = $r->write($buffer, $bufsiz, $offset);
```

- **arg1:** `$r` (`Apache::RequestRec`)
- **arg2:** `$buffer` (scalar)
- **arg3:** `$bufsiz` (scalar)
- **arg4:** `$offset` (number)
- **ret:** `$ret` (number)

16.4 TIE Interface

16.4.1 *OPEN*

META: Autogenerated - needs to be reviewed/completed

```
$ret = OPEN($self, $arg1, $arg2);
```

- **arg1:** `$self` (scalar)
- **arg2:** `$arg1` (scalar)
- **arg3:** `$arg2` (scalar)
- **ret:** `$ret` (integer)

16.4.2 *UNTIE*

META: Autogenerated - needs to be reviewed/completed

```
$ret = $r->UNTIE($refcnt);
```

- **arg1:** `$r` (`Apache::RequestRec`)
- **arg2:** `$refcnt` (`Apache::RequestRec`)
- **ret:** `$ret` (scalar)

16.4.3 *PRINTF*

META: Autogenerated - needs to be reviewed/completed

```
$ret = PRINTF(...);
```

- **arg1:** `...` (scalar)
- **ret:** `$ret` (number)

16.4.4 *CLOSE*

META: Autogenerated - needs to be reviewed/completed

```
$ret = $r->CLOSE();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$ret (scalar)**

16.4.5 PRINT

META: Autogenerated - needs to be reviewed/completed

```
$ret = PRINT(...);
```

- **arg1: ... (scalar)**
- **ret: \$ret (number)**

16.4.6 BINMODE

META: Autogenerated - needs to be reviewed/completed

```
$ret = $r->BINMODE();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$ret (scalar)**

16.4.7 WRITE

META: Autogenerated - needs to be reviewed/completed

```
$ret = $r->WRITE($buffer, $bufsiz, $offset);
```

- **arg1: \$r (Apache::RequestRec)**
- **arg2: \$buffer (scalar)**
- **arg3: \$bufsiz (scalar)**
- **arg4: \$offset (integer)**
- **ret: \$ret (integer)**

16.4.8 TIEHANDLE

META: Autogenerated - needs to be reviewed/completed

```
$ret = TIEHANDLE($stashsv, $sv);
```

- **arg1: \$stashsv (scalar)**
- **arg2: \$sv (scalar)**
- **ret: \$ret (scalar)**

16.4.9 READ

META: Autogenerated - needs to be reviewed/completed

```
$ret = $r->READ($buffer, $len, $offset);
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **arg2:** `$buffer` (scalar)
- **arg3:** `$len` (scalar)
- **arg4:** `$offset` (integer)
- **ret:** `$ret` (scalar)

16.5 See Also

mod_perl 2.0 documentation.

16.6 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

16.7 Authors

The mod_perl development team and numerous contributors.

17 Apache::RequestRec - Perl API for Apache request record accessors

17.1 Synopsis

```
use Apache::RequestRec ();

sub handler{
    my $r = shift;
    ...
    my $auth_type = $r->auth_type;
    ...
    my $s = $r->server;
    my $dir_config = $r->dir_config;
}
```

META: to be completed

17.2 Description

Apache::RequestRec provides the Perl API for Apache request object.

17.3 API

Apache::RequestRec provides the following functions and/or methods:

17.3.1 *proxyreq*

META: Autogenerated - needs to be reviewed/completed

```
$ret = $r->proxyreq($val);
```

- **arg1: \$r (Apache::RequestRec)**
- **arg2: \$val (integer)**
- **ret: \$ret (integer)**

17.3.2 *pool*

META: Autogenerated - needs to be reviewed/completed

The pool associated with the request

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

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$p (APR::Pool)**

17.3.3 connection

META: Autogenerated - needs to be reviewed/completed

The connection record to the client

```
$c = $r->connection();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$c (Apache::Connection)**

17.3.4 server

Get the Apache::Server object for the server the request \$r is running under.

```
$s = $r->server();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$s (Apache::Server)**

17.3.5 next

META: Autogenerated - needs to be reviewed/completed

Pointer to the redirected request if this is an external redirect

```
$next_r = $r->next();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$next_r (Apache::RequestRec)**

17.3.6 prev

META: Autogenerated - needs to be reviewed/completed

Pointer to the previous request if this is an internal redirect

```
$prev_r = $r->prev();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$prev_r (Apache::RequestRec)**

17.3.7 main

Get the main request record

```
$main_r = $r->main();
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **ret:** `$main_r` (**Apache::RequestRec**)

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.

To figure out whether you are inside a main request or a sub-request/internal redirect, use `$r->is_initial_req`.

17.3.8 *the_request*

META: Autogenerated - needs to be reviewed/completed

First line of request

```
$request = $r->the_request();
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **ret:** `$request` (**string**)

17.3.9 *assbackwards*

META: Autogenerated - needs to be reviewed/completed

HTTP/0.9, "simple" request (e.g. GET /foo\n w/no headers)

```
$status = $r->assbackwards($newval);
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **arg2:** `$newval` (**integer**)
- **ret:** `$status` (**integer**)

17.3.10 *header_only*

META: Autogenerated - needs to be reviewed/completed

HEAD request, as opposed to GET

```
$status = $r->header_only();
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **ret:** `$status` (**integer**)

17.3.11 protocol

META: Autogenerated - needs to be reviewed/completed

Protocol string, as given to us, or HTTP/0.9

```
$protocol = $r->protocol();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$protocol (string)**

17.3.12 proto_num

META: Autogenerated - needs to be reviewed/completed

Protocol version number of protocol; 1.1 = 1001

```
$proto_num = $r->proto_num();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$proto_num (integer)**

17.3.13 hostname

META: Autogenerated - needs to be reviewed/completed

Host, as set by full URI or Host:

```
$hostname = $r->hostname();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$hostname (string)**

17.3.14 request_time

META: Autogenerated - needs to be reviewed/completed

Time when the request started

```
$request_time = $r->request_time();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$request_time (number)**

17.3.15 status_line

META: Autogenerated - needs to be reviewed/completed

Status line, if set by script

```
$status_line = $r->status_line();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$status_line (string)**

17.3.16 status

META: Autogenerated - needs to be reviewed/completed

Get/set status line

```
$status = $r->status($new_status);  
$status = $r->status();
```

- **arg1: \$r (Apache::RequestRec)**
- **opt arg2: \$new_status (integer)**

If \$new_status is passed the new status is assigned.

- **ret: \$newval (integer)**

If \$new_status is passed the old status is returned.

17.3.17 method

META: Autogenerated - needs to be reviewed/completed

Request method (eg. GET, HEAD, POST, etc.)

```
$method = $r->method();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$method (string)**

17.3.18 method_number

META: Autogenerated - needs to be reviewed/completed

Apache::M_GET, Apache::M_POST, etc.

```
$methno = $r->method_number();
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **ret:** `$methno` (**integer**)

17.3.19 *allowed*

META: Autogenerated - needs to be reviewed/completed

'allowed' is a bitvector of the allowed methods.

```
$allowed = $r->allowed();
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **ret:** `$allowed` (**number**)

A handler must ensure that the request method is one that it is capable of handling. Generally modules should DECLINE any request methods they do not handle. Prior to aborting the handler like this the handler should set `r->allowed` to the list of methods that it is willing to handle. This bitvector is used to construct the "Allow:" header required for OPTIONS requests, and HTTP_METHOD_NOT_ALLOWED and HTTP_NOT_IMPLEMENTED status codes.

Since the default_handler deals with OPTIONS, all modules can usually decline to deal with OPTIONS. TRACE is always allowed, modules don't need to set it explicitly.

Since the default_handler will always handle a GET, a module which does **not** implement GET should probably return HTTP_METHOD_NOT_ALLOWED. Unfortunately this means that a Script GET handler can't be installed by mod_actions.

17.3.20 *allowed_xmethods*

META: Autogenerated - needs to be reviewed/completed

Array of extension methods

```
$array = $r->allowed_xmethods();
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **ret:** `$array` (**APR::ArrayHeader**)

17.3.21 *allowed_methods*

META: Autogenerated - needs to be reviewed/completed

List of allowed methods

```
$list = $r->allowed_methods();
```

- **arg1:** `$r` (`Apache::RequestRec`)
- **ret:** `$list` (`Apache::MethodList`)

17.3.22 bytes_sent

META: Autogenerated - needs to be reviewed/completed

body byte count, for easy access

```
$bytes_sent = $r->bytes_sent();
```

- **arg1:** `$r` (`Apache::RequestRec`)
- **ret:** `$bytes_sent` (integer)

17.3.23 mtime

META: Autogenerated - needs to be reviewed/completed

Last modified time of the requested resource

```
$mtime = $r->mtime($new_mtime);  
$mtime = $r->mtime();
```

- **arg1:** `$r` (`Apache::RequestRec`)
- **opt arg2:** `$new_mtime` (number)
- **ret:** `$mtime` (number)

17.3.24 remaining

META: Autogenerated - needs to be reviewed/completed

Remaining bytes left to read from the request body

```
$bytes = $r->remaining();
```

META: I think this method is not needed if the deprecated `client_block` methods aren't used, (and plain `$r-<read()` is used instead).

- **arg1:** `$r` (`Apache::RequestRec`)
- **ret:** `$bytes` (integer)

17.3.25 *headers_in*

META: Autogenerated - needs to be reviewed/completed

```
$headers_in = $r->headers_in();
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **ret:** `$headers_in` (**APR::Table**)

17.3.26 *headers_out*

META: Autogenerated - needs to be reviewed/completed

MIME header environment for the response

```
$headers_out = $r->headers_out();
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **ret:** `$headers_out` (**APR::Table**)

17.3.27 *err_headers_out*

META: Autogenerated - needs to be reviewed/completed

MIME header environment for the response, printed even on errors and persist across internal redirects

```
$err_headers_out = $r->err_headers_out();
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **err:** `$err_headers_out` (**APR::Table**)

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).

17.3.28 *notes*

META: Autogenerated - needs to be reviewed/completed

Notes from one module to another

```
$notes = $r->notes();  
$notes = $r->notes($new_notes);
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **opt arg2:** `$new_notes` (**APR::Table**)
- **ret:** `$notes` (**APR::Table**)

The 'notes' is for notes from one module to another, with no other set purpose in mind...

17.3.29 handler

META: Autogenerated - needs to be reviewed/completed

```
$handler = $r->handler();
$handler = $r->handler($new_handler);
```

- **arg1: \$r (Apache::RequestRec)**
- **opt arg2: \$new_handler (string)**
- **ret: \$handler (string)**

17.3.30 content_encoding

META: Autogenerated - needs to be reviewed/completed

```
$ce = $r->content_encoding();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$ce (string)**

17.3.31 content_languages

META: Autogenerated - needs to be reviewed/completed

Array of strings representing the content languages

```
$array_header = $r->content_languages();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$array_header (APR::ArrayHeader)**

17.3.32 user

META: Autogenerated - needs to be reviewed/completed

If an authentication check was made, this gets set to the user name.

```
$r->user($user);
$user = $r->user();
```

- **arg1: \$r (Apache::RequestRec)**
- **opt arg2: \$user (string)**
- **ret: \$user (string)**

17.3.33 *ap_auth_type*

If an authentication check was made, get or set the *ap_auth_type* slot in the request record

```
$auth_type = $r->ap_auth_type();
$r->ap_auth_type($newval);
```

- **arg1: \$r (Apache::RequestRec)**
- **opt arg2: \$newval (string)**

If this argument is passed then a new auth type is assigned. For example:

```
$r->auth_type('Basic');
```

- **ret: \$auth_type (string)**

If \$newval is passed, nothing is returned. Otherwise the current auth type is returned.

ap_auth_type holds the authentication type that has been negotiated between the client and server during the actual request. Generally, *ap_auth_type* is populated automatically when you call `$r->get_basic_auth_pw` so you don't really need to worry too much about it, but if you want to roll your own authentication mechanism then you will have to populate *ap_auth_type* yourself.

Note that `$r->ap_auth_type` was `$r->connection->auth_type` in the mod_perl 1.0 API.

17.3.34 *no_local_copy*

META: Autogenerated - needs to be reviewed/completed

There is no local copy of this response

```
$status = $r->no_local_copy();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$status (integer)**

17.3.35 *unparsed_uri*

META: Autogenerated - needs to be reviewed/completed

The URI without any parsing performed

```
$unparsed_uri = $r->unparsed_uri();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$unparsed_uri (string)**

17.3.36 uri

META: Autogenerated - needs to be reviewed/completed

The path portion of the URI

```
$uri = $r->uri();
$r->uri($uri);
```

- **arg1:** `$r` (`Apache::RequestRec`)
- **opt arg2:** `$uri` (string)
- **ret:** `$uri` (string)

17.3.37 filename

META: Autogenerated - needs to be reviewed/completed

The filename on disk corresponding to this response

```
$filename = $r->filename();
$r->filename($filename);
```

- **arg1:** `$r` (`Apache::RequestRec`)
- **opt arg2:** `$filename` (string)
- **ret:** `$filename` (string)

17.3.38 canonical_filename

META: Autogenerated - needs to be reviewed/completed

```
$canon_filename = $r->canonical_filename();
```

- **arg1:** `$r` (`Apache::RequestRec`)
- **ret:** `$canon_filename` (string)

17.3.39 path_info

META: Autogenerated - needs to be reviewed/completed

The PATH_INFO extracted from this request

```
$path_info = $r->path_info();
$r->path_info($path_info);
```

- **arg1:** `$r` (`Apache::RequestRec`)
- **opt arg2:** `$path_info` (string)
- **ret:** `$path_info` (string)

17.3.40 *args*

META: Autogenerated - needs to be reviewed/completed

The QUERY_ARGS extracted from this request

```
$args = $r->args();
$r->args($args);
```

- **arg1:** \$r (**Apache::RequestRec**)
- **opt arg2:** \$args (string)
- **ret:** \$args (string)

17.3.41 *used_path_info*

META: Autogenerated - needs to be reviewed/completed

Flag for the handler to accept or reject path_info on the current request. All modules should respect the AP_REQ_ACCEPT_PATH_INFO and AP_REQ_REJECT_PATH_INFO values, while AP_REQ_DEFAULT_PATH_INFO indicates they may follow existing conventions. This is set to the user's preference upon HOOK_VERY_FIRST of the fixups.

```
$ret = $r->used_path_info($newval);
```

- **arg1:** \$r (**Apache::RequestRec**)
- **arg2:** \$newval (integer)

17.3.42 *per_dir_config*

META: Autogenerated - needs to be reviewed/completed

These are config vectors, with one void* pointer for each module (the thing pointed to being the module's business). * Options set in config files, etc.

```
$per_dir_config = $r->per_dir_config();
```

- **arg1:** \$r (**Apache::RequestRec**)
- **ret:** \$per_dir_config (**Apache::ConfVector**)

17.3.43 *request_config*

META: Autogenerated - needs to be reviewed/completed

Notes on *this* request

```
$ret = $r->request_config($newval);
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **arg2:** `$newval` (**Apache::ConfVector**)

17.3.44 output_filters

META: Autogenerated - needs to be reviewed/completed

A list of output filters to be used for this request

```
$output_filters = $r->output_filters();
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **ret:** `$output_filters` (**Apache::Filter**)

17.3.45 input_filters

META: Autogenerated - needs to be reviewed/completed

A list of input filters to be used for this request

```
$input_filters = $r->input_filters();
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **ret:** `$input_filters` (**Apache::Filter**)

17.3.46 proto_output_filters

META: Autogenerated - needs to be reviewed/completed

A list of protocol level output filters to be used for this request

```
$proto_output_filters = $r->proto_output_filters();
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **ret:** `$proto_output_filters` (**Apache::Filter**)

17.3.47 proto_input_filters

META: Autogenerated - needs to be reviewed/completed

A list of protocol level input filters to be used for this request

```
$proto_input_filters = $r->proto_input_filters();
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **ret:** `$proto_input_filters` (**Apache::Filter**)

17.4 See Also

mod_perl 2.0 documentation.

17.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

17.6 Authors

The mod_perl development team and numerous contributors.

18 Apache::RequestUtil - Perl API for Apache request record utils

18.1 Synopsis

```
use Apache::RequestUtil ();

# directory level PerlOptions flags lookup
$r->subprocess_env unless $r->is_perl_option_enabled('SetupEnv');
```

META: to be completed

18.2 Description

META: to be completed

18.3 Functions API

18.3.1 *Apache->request()*

```
$request = Apache->request;
```

18.4 Methods API

18.4.1 *default_type*

META: Autogenerated - needs to be reviewed/completed

Retrieve the value of the DefaultType directive, or text/plain if not set

```
$ret = $r->default_type();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (string)**

The default type

18.4.2 *document_root*

META: Autogenerated - needs to be reviewed/completed

Retrieve the document root for this server

```
$ret = $r->document_root();
```

18.4.3 `get_limit_req_body`

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (string)**

The document root

18.4.3 `get_limit_req_body`

META: Autogenerated - needs to be reviewed/completed

Return the limit on bytes in request msg body

```
$ret = $r->get_limit_req_body();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

the maximum number of bytes in the request msg body

18.4.4 `get_server_name`

META: Autogenerated - needs to be reviewed/completed

Get the current server name from the request

```
$ret = $r->get_server_name();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (string)**

the server name

18.4.5 `get_server_port`

META: Autogenerated - needs to be reviewed/completed

Get the current server port

```
$ret = $r->get_server_port();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: \$ret (integer)**

The server's port

18.4.6 is_initial_req

META: Autogenerated - needs to be reviewed/completed

Determine if the current request is the main request or a sub requests

```
$ret = $r->is_initial_req();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

18.4.7 add_config

META: Autogenerated - needs to be reviewed/completed

```
$ret = $r->add_config($lines, $path, $override);
```

- **arg1: \$r (Apache::RequestRec)**
- **arg2: \$lines (ARRAY ref)**
- **opt arg3: \$path (scalar)**
- **opt arg4: \$override (string)**
- **ret: \$ret (string)**

18.4.8 location

META: Autogenerated - needs to be reviewed/completed

```
$location = $r->location($location);
```

- **arg1: \$r (Apache::RequestRec)**
- **opt arg2: \$location (string)**
- **ret: \$location (integer)**

18.4.9 location_merge

META: Autogenerated - needs to be reviewed/completed

```
$ret = $r->location_merge($location);
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **arg2:** `$location` (**string**)
- **ret:** `$ret` (**integer**)

18.4.10 pnotes

META: Autogenerated - needs to be reviewed/completed

Notes from one module to another

```
$pnotes = $r->pnotes();  
$pnotes = $r->pnotes($new_pnotes);
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **opt arg2:** `$new_pnotes` (**APR::Table**)
- **ret:** `$pnotes` (**APR::Table**)

Similar to (**Apache::RequestRec**), but values can be any perl variables. That also means that it can be used only between perl modules.

18.4.11 no_cache

META: Autogenerated - needs to be reviewed/completed

```
$ret = $r->no_cache($flag);
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **arg2:** `$flag` (**number**)
- **ret:** `$ret` (**integer**)

18.4.12 as_string

META: Autogenerated - needs to be reviewed/completed

```
$string = $r->as_string();
```

- **arg1:** `$r` (**Apache::RequestRec**)
- **ret:** `$string` (**string**)

18.4.13 get_handlers

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

```
@handlers = $r->get_handlers($hook_name);
```

- **arg1: \$r (Apache::RequestRec)**
- **arg2: \$hook_name (string)**

a string representing the phase to handle.

- **ret: @handlers (CODE ref or ref to ARRAY of CODE refs)**

a list of references to the handler subroutines

For example:

```
@handlers = $r->get_handlers('PerlResponseHandler');
```

18.4.14 push_handlers

META: Autogenerated - needs to be reviewed/completed

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

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

- **arg1: \$r (Apache::RequestRec)**
- **arg2: \$hook_name (string)**

a string representing the phase to handle.

- **arg3: \$handlers (CODE ref or ref to ARRAY of CODE refs)**

a reference to a list of references to the handler subroutines, or a single reference to a handler subroutine

- **ret: no return value**

Examples:

```
$r->push_handlers(PerlResponseHandler => \&handler);
$r->push_handlers(PerlResponseHandler => [\&handler, \&handler2]);

# XXX: not implemented yet
$r->push_handlers(PerlResponseHandler => sub {...});
```

18.4.15 set_handlers

META: Autogenerated - needs to be reviewed/completed

Set a list of handlers to be called for a given phase.

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

- **arg1: \$r (Apache::RequestRec)**
- **arg2: \$hook_name (string)**

a string representing the phase to handle.

- **arg3: \$handlers (CODE ref or ref to ARRAY of CODE refs)**

a reference to a list of references to the handler subroutines, or a single reference to a handler subroutine

- **ret: no return value**

Examples:

```
$r->set_handlers(PerlResponseHandler => \&handler);
$r->set_handlers(PerlResponseHandler => [\&handler, \&handler2]);

# XXX: not implemented yet
$r->set_handlers(PerlResponseHandler => sub {...});
```

18.4.16 set_basic_credentials

META: Autogenerated - needs to be reviewed/completed

```
$r->set_basic_credentials($username, $password);
```

- **arg1: \$r (Apache::RequestRec)**
- **arg2: \$username (string)**
- **arg3: \$password (string)**
- **ret: no return value**

18.4.17 slurp_filename

META: Autogenerated - needs to be reviewed/completed

Return a reference to contents of \$r->filename.

```
$content = $r->slurp_filename($tainted);
```

- **arg1: \$r (Apache::RequestRec)**
- **arg2: \$tainted (number)**

By default the returned data is tainted (if run under `-T`). If an optional `$tainted` flag is set to zero, the data will be marked as non-tainted. Do not set this flag to zero unless you know what you are doing, you may create a security hole in your program if you do. For more information see the *perlsec* manpage. If you wonder why this option is available, it is used internally by the `ModPerl::Registry` handler and friends, because the CGI scripts that it reads are considered safe (you could just as well `require()` them).

- **ret: \$content** (scalar)

18.4.18 is_perl_option_enabled

check whether a directory level PerlOptions flag is enabled or not.

```
$result = $r->is_perl_option_enabled($flag);
```

- **arg1: \$r** (`Apache::RequestRec`)
- **arg2: \$flag** (string)
- **ret: \$result** (integer)

For example to check whether the `SetupEnv` option is enabled for the current request (which can be disabled with `PerlOptions -SetupEnv`) and populate the environment variables table if disabled:

```
$r->subprocess_env unless $r->is_perl_option_enabled('SetupEnv');
```

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

18.4.19 dir_config

`dir_config()` provides an interface for the per-directory variable specified by the `PerlSetVar` and `PerlAddVar` directives, and also can be manipulated via the `APR::Table` methods.

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

- **arg1: \$r** (`Apache::RequestRec`)
- **opt arg2: \$key** (string)
- **opt arg3: \$val** (string)
- **ret: \$ret** (scalar)

Depends on the passed arguments, see further discussion

The keys are case-insensitive.

```
$apr_table = $r->dir_config();
```

`dir_config()` called in a scalar context without the `$key` argument returns a *HASH* reference blessed into the `APR::Table` class. This object can be manipulated via the `APR::Table` methods. For available methods see the `APR::Table` manpage.

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

If the `$key` argument is passed in the list context a list of all matching values will be returned. This method is ineffective for big tables, as it does a linear search of the table. Therefore avoid using this way of calling `dir_config()` unless you know that there could be more than one value for the wanted key and all the values are wanted.

```
$value = $r->dir_config($key);
```

If the `$key` argument is passed in the scalar context only a single value will be returned. Since the table preserves the insertion order, if there is more than one value for the same key, the oldest value associated with the desired key is returned. Calling in the scalar context is also much faster, as it'll stop searching the table as soon as the first match happens.

```
$r->dir_config($key => $val);
```

If the `$key` and the `$val` arguments are used, the `set()` operation will happen: all existing values associated with the key `$key` (and the key itself) will be deleted and `$value` will be placed instead.

```
$r->dir_config($key => undef);
```

If `$val` is *undef* the `unset()` operation will happen: all existing values associated with the key `$key` (and the key itself) will be deleted.

18.5 See Also

`mod_perl 2.0` documentation.

18.6 Copyright

`mod_perl 2.0` and its core modules are copyrighted under The Apache Software License, Version 1.1.

18.7 Authors

The `mod_perl` development team and numerous contributors.

19 Apache::Response - Perl API for Apache HTTP request response methods

19.1 Synopsis

```
use Apache::Response ();
```

META: to be completed

19.2 Description

META: to be completed

19.3 API

`Apache::Response` provides the following functions and/or methods:

19.3.1 custom_response

META: Autogenerated - needs to be reviewed/completed

Install a custom response handler for a given status

```
$r->custom_response($status, $string);
```

- **arg1: \$r (`Apache::RequestRec`)**

The current request

- **arg2: \$status (integer)**

The status for which the custom response should be used

- **arg3: \$string (string)**

The custom response. This can be a static string, a file or a URL

- **ret: no return value**

19.3.2 make_etag

META: Autogenerated - needs to be reviewed/completed

Construct an entity tag from the resource information. If it's a real file, build in some of the file characteristics.

```
$etag = $r->make_etag($force_weak);
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **arg2: \$force_weak (number)**

Force the entity tag to be weak - it could be modified again in as short an interval.

- **ret: \$etag (string)**

The entity tag

19.3.3 meets_conditions

META: Autogenerated - needs to be reviewed/completed

Implements condition GET rules for HTTP/1.1 specification. This function inspects the client headers and determines if the response fulfills the requirements specified.

```
$ret = $r->meets_conditions();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

1 if the response fulfills the condition GET rules, 0 otherwise

19.3.4 rationalize_mtime

META: Autogenerated - needs to be reviewed/completed

Return the latest rational time from a request/mtime pair. Mtime is returned unless it's in the future, in which case we return the current time.

```
$rat_mtime = $r->rationalize_mtime($mtime);
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **arg2: \$mtime (number)**

The last modified time

- **ret: \$rat_mtime (number)**

the latest rational time.

19.3.5 send_error_response

META: Autogenerated - needs to be reviewed/completed

Send error back to client.

```
$r->send_error_response($recursive_error);
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **arg2: \$recursive_error (string)**

last arg indicates error status in case we get an error in the process of trying to deal with an `ErrorDocument` to handle some other error. In that case, we print the default report for the first thing that went wrong, and more briefly report on the problem with the `ErrorDocument`.

- **ret: no return value**

19.3.6 send_mmap

META: Autogenerated - needs to be reviewed/completed

Send an MMAP'ed file to the client

```
$ret = $r->send_mmap($mm, $offset, $length);
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **arg2: \$mm (APR::Mmap)**

The MMAP'ed file to send

- **arg3: \$offset (number)**

The offset into the MMAP to start sending

- **arg4: \$length (integer)**

The amount of data to send

- **ret: \$ret (integer)**

The number of bytes sent

19.3.7 set_content_length

META: Autogenerated - needs to be reviewed/completed

Set the content length for this request.

```
$r->set_content_length($length);
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **arg2: \$length (integer)**

The new content length

- **ret: no return value**

19.3.8 set_etag

META: Autogenerated - needs to be reviewed/completed

Set the E-tag outgoing header

```
$r->set_etag();
```

- **arg1: \$r (Apache::RequestRec)**
- **ret: no return value**

19.3.9 set_keepalive

META: Autogenerated - needs to be reviewed/completed

Set the keepalive status for this request

```
$ret = $r->set_keepalive();
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **ret: \$ret (integer)**

1 if keepalive can be set, 0 otherwise

19.3.10 update_mtime

META: Autogenerated - needs to be reviewed/completed

Function to set the `r->mtime` field to the specified value if it's later than what's already there.

```
$r->update_mtime($dependency_mtime);
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **arg2: \$dependency_mtime (number)**
- **ret: no return value**

19.3.11 set_last_modified

META: Autogenerated - needs to be reviewed/completed

```
$r->set_last_modified($mtime);
```

- **arg1: \$r (Apache::RequestRec)**
- **arg2: \$mtime (number)**
- **ret: no return value**

19.3.12 send_cgi_header

META: Autogenerated - needs to be reviewed/completed

```
$r->send_cgi_header($buffer);
```

- **arg1: \$r (Apache::RequestRec)**
- **arg2: \$buffer (string)**
- **ret: no return value**

19.4 See Also

`mod_perl 2.0` documentation.

19.5 Copyright

`mod_perl 2.0` and its core modules are copyrighted under The Apache Software License, Version 1.1.

19.6 Authors

The mod_perl development team and numerous contributors.

20 Apache::Server - Perl API for for Apache server record accessors

20.1 Synopsis

```
use Apache::Server ();
```

META: to be completed

20.2 Description

META: to be completed

20.3 API

Apache::Server provides the following functions and/or methods:

20.3.1 *process*

META: Autogenerated - needs to be reviewed/completed

The process this server is running in

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

- **arg1: \$s (Apache::Server)**
- **ret: \$proc (Apache::Process)**

20.3.2 *next*

META: Autogenerated - needs to be reviewed/completed

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

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

- **arg1: \$s (Apache::Server)**
- **ret: \$next_s (Apache::Server)**

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

```
use Apache::Server ();
use Apache::ServerUtil ();
my $server = Apache->server;
my $vhosts = 0;
for (my $s = $server->next; $s; $s = $s->next) {
    $vhosts++;
}
```

20.3.3 *server_admin*

Get/set the server admin value

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_server_admin (string)**

If passed, sets the new server_admin.

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

Returns the server_admin setting.

If \$new_server_admin is passed returns the setting before the change.

20.3.4 *server_hostname*

Get/set the server hostname value

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_server_hostname (string)**

If passed, sets the new server_hostname.

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

Returns the server_hostname setting.

If \$new_server_hostname is passed returns the setting before the change.

20.3.5 *port*

META: Autogenerated - needs to be reviewed/completed

Get/set the port value

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_port (string)**

If passed, sets the new port.

- **ret: \$port (string)**

Returns the port setting.

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

20.3.6 *error_fname*

META: Autogenerated - needs to be reviewed/completed

Get/set the error_fname value

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_error_fname (string)**

If passed, sets the new error_fname.

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

Returns the error_fname setting.

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

20.3.7 *loglevel*

META: Autogenerated - needs to be reviewed/completed

Get/set the log level value

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_loglevel (string)**

If passed, sets the new loglevel.

- **ret: \$loglevel (string)**

Returns the loglevel setting.

If \$new_loglevel is passed returns the setting before the change.

20.3.8 *is_virtual*

META: Autogenerated - needs to be reviewed/completed

Get/set the is_virtual value

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_is_virtual (string)**

If passed, sets the new is_virtual.

META: this is wrong, it should be a read only accessor

- **ret: \$is_virtual (string)**

Returns the is_virtual setting.

If \$new_is_virtual is passed returns the setting before the change.

20.3.9 *module_config*

META: Autogenerated - needs to be reviewed/completed

Get/set config vector containing pointers to modules' per-server config structures.

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: new_module_config (Apache::ConfVector)**

If passed, sets the new module_config.

- **ret: \$module_config (Apache::ConfVector)**

Returns the module_config setting.

If \$new_module_config is passed returns the setting before the change.

20.3.10 *lookup_defaults*

META: Autogenerated - needs to be reviewed/completed

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

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_lookup_defaults (Apache::ConfVector)**

If passed, sets the new lookup_defaults.

- **ret: \$lookup_defaults (Apache::ConfVector)**

Returns the lookup_defaults setting.

If \$new_lookup_defaults is passed returns the setting before the change.

20.3.11 *addr*s

META: Autogenerated - needs to be reviewed/completed

Get/set the *addr*s value

```
$addr = $s->addr();
$prev_addr = $s->addr($new_addr);
```

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_addr (Apache::ServerAddr)**

If passed, sets the new *addr*s.

- **ret: \$addr (Apache::ServerAddr)**

Returns the *addr*s setting.

If \$new_addr is passed returns the setting before the change.

20.3.12 *timeout*

META: Autogenerated - needs to be reviewed/completed

Get/set the timeout, as an apr interval, before we give up

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_timeout (string)**

If passed, sets the new timeout.

- **ret: \$timeout (string)**

Returns the timeout setting.

If \$new_timeout is passed returns the setting before the change.

20.3.13 keep_alive_timeout

META: Autogenerated - needs to be reviewed/completed

Get/set the apr interval we will wait for another request

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_keep_alive_timeout (string)**

If passed, sets the new keep_alive_timeout.

- **ret: \$keep_alive_timeout (string)**

Returns the keep_alive_timeout setting.

If \$new_keep_alive_timeout is passed returns the setting before the change.

20.3.14 keep_alive_max

META: Autogenerated - needs to be reviewed/completed

Get/set maximum requests per connection

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_keep_alive_max (string)**

If passed, sets the new keep_alive_max.

- **ret: \$keep_alive_max (string)**

Returns the keep_alive_max setting.

If \$new_keep_alive_max is passed returns the setting before the change.

20.3.15 *keep_alive*

META: Autogenerated - needs to be reviewed/completed

Use persistent connections?

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_keep_alive (string)**

If passed, sets the new keep_alive.

- **ret: \$keep_alive (string)**

Returns the keep_alive setting.

If \$new_keep_alive is passed returns the setting before the change.

20.3.16 *path*

META: Autogenerated - needs to be reviewed/completed

Get/set pathname for ServerPath

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_path (string)**

If passed, sets the new path.

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

Returns the path setting.

If \$new_path is passed returns the setting before the change.

20.3.17 *names*

META: Autogenerated - needs to be reviewed/completed

Get/set normal names for ServerAlias servers

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_names (APR::ArrayHeader)**

If passed, sets the new names.

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

Returns the names setting.

If \$new_names is passed returns the setting before the change.

20.3.18 wild_names

META: Autogenerated - needs to be reviewed/completed

Wildcarded names for ServerAlias servers

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_wild_names (APR::ArrayHeader)**

If passed, sets the new wild_names.

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

Returns the wild_names setting.

If \$new_wild_names is passed returns the setting before the change.

20.3.19 limit_req_line

META: Autogenerated - needs to be reviewed/completed

Get/set limit on size of the HTTP request line

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_limit_req_line (string)**

If passed, sets the new limit_req_line.

- **ret: \$limit_req_line (string)**

Returns the limit_req_line setting.

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

20.3.20 *limit_req_fieldsize*

META: Autogenerated - needs to be reviewed/completed

limit on size of any request header field

```
$limit_req_fieldsize = $s->limit_req_fieldsize();
$prev_limit_req_fieldsize = $s->limit_req_fieldsize($new_limit_req_fieldsize);
```

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_limit_req_fieldsize (string)**

If passed, sets the new `limit_req_fieldsize`.

- **ret: \$limit_req_fieldsize (string)**

Returns the `limit_req_fieldsize` setting.

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

20.3.21 *limit_req_fields*

META: Autogenerated - needs to be reviewed/completed

Get/set limit on number of request header fields

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$new_limit_req_fields (string)**

If passed, sets the new `limit_req_fields`.

- **ret: \$limit_req_fields (string)**

Returns the `limit_req_fields` setting.

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

20.4 See Also

mod_perl 2.0 documentation.

20.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

20.6 Authors

The mod_perl development team and numerous contributors.

21 Apache::ServerUtil - Perl API for XXX

21.1 Synopsis

```
use Apache::ServerUtil ();

$s = Apache->server;
my $srv_cfg = $s->dir_config;

# get 'conf/' dir path using $s
my $conf_dir = $s->server_root_relative('conf');

# server level PerlOptions flags lookup
$s->push_handlers(ChildExit => \&child_exit)
    if $s->is_perl_option_enabled('ChildExit');
```

META: to be completed

21.2 Description

`Apache::ServerUtil` provides the Perl API for Apache server object.

META: to be completed

21.3 Constants

21.3.1 `Apache::Server::server_root`

returns the value set by the `ServerRoot` directive.

21.4 Functions API

21.4.1 `add_version_component`

META: Autogenerated - needs to be reviewed/completed

Add a component to the version string

```
add_version_component($pconf_pool, $component);
```

- **arg1: `$pconf` (`APR::Pool`)**

The pool to allocate the component from (should really be a `$pconf_pool`)

- **arg2: `$component` (string)**

The string to add

- **ret: no return value**

21.4.2 *exists_config_define*

Check for a definition from the server command line

```
$result = Apache::Server::exists_config_define($name);
```

- **arg1: \$name (string)**

The define to check for

- **ret: \$result (integer)**

true if defined, false otherwise

For example:

```
print "this is mp2" if Apache::Server::exists_config_define('MODPERL2');
```

21.4.3 *get_server_built*

META: Autogenerated - needs to be reviewed/completed

Get the date and time that the server was built

```
$when_built = Apache::Server::get_server_built();
```

- **ret: \$when_built (string)**

The server build time string

21.4.4 *get_server_version*

Get the server version string

```
Apache::Server::get_server_version();
```

- **ret: \$ret (string)**

The server version string

21.5 Methods API

Apache::ServerUtil provides the following functions and/or methods:

21.5.1 *server_root_relative()*

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

```
$path = $s->server_root_relative($fname);
```

- **arg1: \$s** (**Apache::Server**)
- **opt arg2: \$fname** (string)
- **ret: \$path** (string)

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

```
my $log_dir = Apache::Server::server_root_relative($r->pool, 'logs');
```

If `$fname` is not specified, the value of `ServerRoot` is returned with a trailing `/`. (it's the same as using `' '` as `$fname`'s value).

Also see the `Apache::Server::server_root` constant.

21.5.2 *error_log2stderr*

META: Autogenerated - needs to be reviewed/completed

Convert stderr to the error log

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

- **arg1: \$s** (**Apache::Server**)

The current server

- **ret: no return value**

21.5.3 *psignature*

META: Autogenerated - needs to be reviewed/completed

Get HTML describing the address and (optionally) admin of the server.

```
$sig = $r->psignature($prefix);
```

- **arg1: \$r** (**Apache::RequestRec**)
- **arg2: \$prefix** (string)

Text which is prepended to the return value

- **ret: \$sig** (string)

HTML describing the server

21.5.4 *dir_config*

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

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

- **arg1: \$s (Apache::Server)**
- **opt arg2: \$key (string)**
- **opt arg3: \$val (string)**
- **ret: \$ret (scalar)**

Depends on the passed arguments, see further discussion

The keys are case-insensitive.

```
$t = $s->dir_config();
```

`dir_config()` called in a scalar context without the `$key` argument returns a *HASH* reference blessed into the *APR::Table* class. This object can be manipulated via the *APR::Table* methods. For available methods see *APR::Table*.

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

If the `$key` argument is passed in the list context a list of all matching values will be returned. This method is ineffective for big tables, as it does a linear search of the table. Therefore avoid using this way of calling `dir_config()` unless you know that there could be more than one value for the wanted key and all the values are wanted.

```
$value = $s->dir_config($key);
```

If the `$key` argument is passed in the scalar context only a single value will be returned. Since the table preserves the insertion order, if there is more than one value for the same key, the oldest value associated with the desired key is returned. Calling in the scalar context is also much faster, as it'll stop searching the table as soon as the first match happens.

```
$s->dir_config($key => $val);
```

If the `$key` and the `$val` arguments are used, the `set()` operation will happen: all existing values associated with the key `$key` (and the key itself) will be deleted and `$value` will be placed instead.

```
$s->dir_config($key => undef);
```

If `$val` is *undef* the `unset()` operation will happen: all existing values associated with the key `$key` (and the key itself) will be deleted.

21.5.5 *is_perl_option_enabled*

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

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

- **arg1: `$s` (Apache::Server)**
- **arg2: `$flag` (string)**
- **ret: `$result` (integer)**

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

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

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

21.5.6 *get_handlers*

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

```
@handlers = $s->get_handlers($hook_name);
```

- **arg1: `$s` (Apache::Server)**
- **arg2: `$hook_name` (string)**
a string representing the phase to handle.
- **ret: `@handlers` (CODE ref or ref to ARRAY of CODE refs)**
a list of references to the handler subroutines

For example:

```
@handlers = $s->get_handlers('PerlResponseHandler');
```

21.5.7 *push_handlers*

META: Autogenerated - needs to be reviewed/completed

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


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

- **arg1: \$s (Apache::Server)**
- **arg2: \$hook_name (string)**

a string representing the phase to handle.

- **arg3: \$handlers (CODE ref or ref to ARRAY of CODE refs)**

a reference to a list of references to the handler subroutines, or a single reference to a handler subroutine

- **ret: no return value**

Examples:

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

# XXX: not implemented yet
$s->push_handlers(PerlResponseHandler => sub {...});
```

21.5.8 set_handlers

META: Autogenerated - needs to be reviewed/completed

Set a list of handlers to be called for a given phase.

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

- **arg1: \$s (Apache::Server)**
- **arg2: \$hook_name (string)**

a string representing the phase to handle.

- **arg3: \$handlers (CODE ref or ref to ARRAY of CODE refs)**

a reference to a list of references to the handler subroutines, or a single reference to a handler subroutine

- **ret: no return value**

Examples:

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

```
# XXX: not implemented yet
$s->set_handlers(PerlResponseHandler => sub {...});
```

21.5.9 *method_register*

META: Autogenerated - needs to be reviewed/completed

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

```
$ret = $p->method_register($methname);
```

- **arg1: \$p (APR::Pool)**

The pool to create registered method numbers from.

- **arg2: \$methname (string)**

The name of the new method to register.

- **ret: \$ret (integer)**

An int value representing an offset into a bitmask.

21.5.10 *get_status_line*

META: Autogenerated - needs to be reviewed/completed

Return the Status-Line for a given status code (excluding the HTTP-Version field). If an invalid or unknown status code is passed, "500 Internal Server Error" will be returned.

```
$ret = get_status_line($status);
```

- **arg1: \$status (integer)**

The HTTP status code

- **ret: \$ret (string)**

The Status-Line

21.5.11 *server*

Get the main server's object

```
$main_s = Apache->server();
```

- **arg1: Apache (class name)**
- **ret: \$main_s (Apache::Server)**

21.6 See Also

mod_perl 2.0 documentation.

21.7 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

21.8 Authors

The mod_perl development team and numerous contributors.

22 Apache::SubProcess -- Executing SubProcesses from mod_perl

22.1 Synopsis

```

use Apache::SubProcess ();

use Config;
use constant PERLIO_IS_ENABLED => $Config{useperlio};

# pass @ARGV / read from the process
$command = "/tmp/argv.pl";
@argv = qw(foo bar);
$out_fh = Apache::SubProcess::spawn_proc_prog($r, $command, \@argv);
$output = read_data($out_fh);

# pass environment / read from the process
$command = "/tmp/env.pl";
$r->subprocess_env->set(foo => "bar");
$out_fh = Apache::SubProcess::spawn_proc_prog($r, $command);
$output = read_data($out_fh);

# write to/read from the process
$command = "/tmp/in_out_err.pl";
($in_fh, $out_fh, $err_fh) =
    Apache::SubProcess::spawn_proc_prog($r, $command);
print $in_fh "hello\n";
$output = read_data($out_fh);
$error = read_data($err_fh);

# helper function to work w/ and w/o perlio-enabled Perl
sub read_data {
    my($fh) = @_;
    my $data;
    if (PERLIO_IS_ENABLED || IO::Select->new($fh)->can_read(10)) {
        $data = <$fh>;
    }
    return defined $data ? $data : '';
}

```

22.2 Description

Apache::SubProcess provides the Perl API for running and communicating with processes spawned from mod_perl handlers.

22.3 API

22.3.1 *spawn_proc_prog*

```

$out_fh =
    Apache::SubProcess::spawn_proc_prog($r, $command, [\@argv]);
($in_fh, $out_fh, $err_fh) =
    Apache::SubProcess::spawn_proc_prog($r, $command, [\@argv]);

```

22.4 See Also

`spawn_proc_prog()` spawns a sub-process which `exec()`'s `$command` and returns the output pipe filehandle in the scalar context, or input, output and error pipe filehandles in the list context. Using these three pipes it's possible to communicate with the spawned process.

The third optional argument is a reference to an array which if passed becomes `ARGV` to the spawned program.

It's possible to pass environment variables as well, by calling:

```
$r->subprocess_env->set($key => $value);
```

before spawning the subprocess.

There is an issue with reading from the read filehandle (`$in_fh`):

A pipe filehandle returned under `perlio-disabled` Perl needs to call `select()` if the other end is not fast enough to send the data, since the read is non-blocking.

A pipe filehandle returned under `perlio-enabled` Perl on the other hand does the `select()` internally, because it's really a filehandle opened via `:APR` layer, which internally uses `APR` to communicate with the pipe. The way `APR` is implemented Perl's `select()` cannot be used with it (mainly because `select()` wants `fileno()` and `APR` is a crossplatform implementation which hides the internal datastructure).

Therefore to write a portable code, you want to use `select` for `perlio-disabled` Perl and do nothing for `perlio-enabled` Perl, hence you can use something similar to the `read_data()` wrapper shown in the `SYNOPSIS` section.

22.4 See Also

`mod_perl 2.0` documentation.

22.5 Copyright

`mod_perl 2.0` and its core modules are copyrighted under The Apache Software License, Version 1.1.

22.6 Authors

The `mod_perl` development team and numerous contributors.

23 Apache::SubRequest - Perl API for Apache subrequests

23.1 Synopsis

```
use Apache::SubRequest ( );
```

META: to be completed

23.2 Description

META: to be completed

23.3 API

`Apache::SubRequest` provides the following functions and/or methods:

23.3.1 *DESTROY*

META: Autogenerated - needs to be reviewed/completed

Free the memory associated with a sub request

```
$r->DESTROY( );
```

- **arg1: \$r (`Apache::RequestRec`)**

The sub request to finish

- **ret: no return value**

23.3.2 *internal_fast_redirect*

META: Autogenerated - needs to be reviewed/completed

Redirect the current request to a `sub_req`, merging the pools

```
$r->internal_fast_redirect($sub_req);
```

- **arg1: \$r (`Apache::RequestRec`)**

The current request

- **arg2: \$sub_req (string)**

A subrequest created from this request

- **ret: no return value**

23.3.3 *internal_redirect*

META: Autogenerated - needs to be reviewed/completed

Then there's the case that you want some other request to be served as the top-level request **INSTEAD** of what the client requested directly. If so, call this from a handler, and then immediately return OK.

Redirect the current request to some other uri

```
$r->internal_redirect($new_uri);
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **arg2: \$new_uri (string)**

The URI to replace the current request with

- **ret: no return value**

23.3.4 *internal_redirect_handler*

META: Autogenerated - needs to be reviewed/completed

This function is designed for things like actions or CGI scripts, when using AddHandler, and you want to preserve the content type across an internal redirect.

```
$r->internal_redirect_handler($new_uri);
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **arg2: \$new_uri (string)**

The URI to replace the current request with.

- **ret: no return value**

23.3.5 *lookup_dirent*

META: Autogenerated - needs to be reviewed/completed

Create a sub request for the given apr_dir_read result. This sub request can be inspected to find information about the requested file

```
$lr = $r->lookup_dirent($finfo, $subtype, $next_filter);
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **arg2: \$finfo (APR::Finfo)**

The apr_dir_read result to lookup

- **arg3: \$subtype (integer)**

What type of subrequest to perform, one of;

```
Apache::SUBREQ_NO_ARGS      ignore r->args and r->path_info
Apache::SUBREQ_MERGE_ARGS   merge  r->args and r->path_info
```

- **arg4: \$next_filter (integer)**

The first filter the sub_request should use. If this is NULL, it defaults to the first filter for the main request

- **ret: \$lr (Apache::RequestRec)**

The new request record

23.3.6 *lookup_file*

META: Autogenerated - needs to be reviewed/completed

Create a sub request for the given file. This sub request can be inspected to find information about the requested file

```
$ret = $r->lookup_file($new_file, $next_filter);
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **arg2: \$new_file (string)**

The URI to lookup

- **arg3: \$next_filter (Apache::Filter)**

The first filter the sub_request should use. If this is NULL, it defaults to the first filter for the main request

- **ret: \$ret (Apache::SubRequest)**

The new request record

23.3.7 *lookup_uri*

META: Autogenerated - needs to be reviewed/completed

Create a sub request from the given URI. This sub request can be inspected to find information about the requested URI.

```
$ret = $r->lookup_uri($new_file, $next_filter);
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **arg2: \$new_file (string)**

The URI to lookup

- **arg3: \$next_filter (Apache::Filter)**

The first filter the sub_request should use. If this is NULL, it defaults to the first filter for the main request

- **ret: \$ret (Apache::SubRequest)**

The new request record

23.3.8 *lookup_method_uri*

META: Autogenerated - needs to be reviewed/completed

Create a sub request for the given URI using a specific method. This sub request can be inspected to find information about the requested URI

```
$ret = $r->lookup_method_uri($method, $new_file, $next_filter);
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **arg2: \$method (string)**

The method to use in the new sub request

- **arg3: \$new_file (string)**

The URI to lookup

- **arg4: \$next_filter (Apache::Filter)**

The first filter the sub_request should use. If this is NULL, it defaults to the first filter for the main request

- **ret: \$ret (Apache::SubRequest)**

The new request record

23.4 See Also

mod_perl 2.0 documentation.

23.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

23.6 Authors

The mod_perl development team and numerous contributors.

24 Apache::URI - Perl API for manipulating URIs

24.1 Synopsis

```
use Apache::URI ();
```

META: to be completed

24.2 Description

META: to be completed

24.3 API

Apache::URI provides the following functions and/or methods:

24.3.1 *construct_server*

META: Autogenerated - needs to be reviewed/completed

Construct a full hostname

```
$hostname = $r->construct_server($hostname, $port, $p);
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **arg2: \$hostname (string)**

The hostname of the server

- **arg3: \$port (string)**

The port the server is running on

- **arg4: \$p (APR::Pool)**

The pool to allocate from

- **ret: \$hostname (string)**

The server's hostname

24.3.2 *construct_url*

META: Autogenerated - needs to be reviewed/completed

build a fully qualified URL from the uri and information in the request rec

```
$ret = $r->construct_url($uri, $p);
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **arg2: \$uri (string)**

The path to the requested file

- **arg3: \$p (APR::Pool)**

The pool to allocate the URL from

- **ret: \$ret (string)**

A fully qualified URL

24.3.3 *parse_uri*

META: Autogenerated - needs to be reviewed/completed

parse_uri: break apart the uri

```
$r->parse_uri($uri);
```

- **arg1: \$r (Apache::RequestRec)**

The current request

- **arg2: \$uri (string)**

The uri to break apart

- **ret: no return value**

24.4 See Also

mod_perl 2.0 documentation.

24.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

24.6 Authors

The mod_perl development team and numerous contributors.

25 Apache::Util - Perl API for XXX

25.1 Synopsis

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

META: to be completed

25.2 Description

META: to be completed

25.3 Functions API

`Apache::Util` provides the following functions and/or methods:

25.3.1 *format_time*

META: Autogenerated - needs to be reviewed/completed

Convert a time from an integer into a string in a specified format

```
$time_str = Apache::format_time($time, $fmt, $gmt, $p);
```

- **arg1: \$time (number)**

The time to convert

- **arg2: \$fmt (string)**

The format to use for the conversion

- **arg3: \$gmt (integer)**

Convert the time for GMT?

- **arg4: \$p (`APR::Pool`)**

The pool to allocate memory from

- **ret: \$time_str (string)**

The string that represents the specified time

25.3.2 *escape_path*

META: Autogenerated - needs to be reviewed/completed

convert an OS path to a URL in an OS dependant way.

```
$path = Apache::escape_path($path, $p, $partial);
```

- **arg1: \$path (string)**

The path to convert

- **arg2: \$p (APR::Pool)**

The pool to allocate from

- **arg3: \$partial (integer)**

if set, assume that the path will be appended to something with a '/' in it (and thus does not prefix "/")

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

The converted URL

25.4 See Also

mod_perl 2.0 documentation.

25.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

25.6 Authors

The mod_perl development team and numerous contributors.

26 APR - Perl Interface for libapr and libaprutil Libraries

26.1 Synopsis

```
use APR;
```

26.2 Description

Notes on how to use APR outside mod_perl 2.0.

Normally you don't need to use this module. However if you are using an `APR::` package outside of mod_perl, you need to load APR first. For example:

```
% perl -MApache2 -MAPR -MAPR::UUID -le 'print APR::UUID->new->format'
```

26.3 See Also

mod_perl 2.0 documentation.

26.4 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

26.5 Authors

The mod_perl development team and numerous contributors.

27 APR::Base64 - Perl API for XXX

27.1 Synopsis

```
use APR::Base64 ();
```

META: to be completed

27.2 Description

META: to be completed

27.3 API

APR::Base64 provides the following functions and/or methods:

27.3.1 *encode_len*

META: Autogenerated - needs to be reviewed/completed

```
$ret = encode_len($len);
```

- **arg1: \$len (integer)**

the length of an unencrypted string.

- **ret: \$ret (integer)**

the length of the string after it is encrypted

27.4 See Also

mod_perl 2.0 documentation.

27.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

27.6 Authors

The mod_perl development team and numerous contributors.

28 APR::Brigade - Perl API for XXX

28.1 Synopsis

```
use APR::Brigade ();
```

META: to be completed

28.2 Description

META: to be completed

28.3 API

APR::Brigade provides the following functions and/or methods:

28.3.1 *destroy*

META: Autogenerated - needs to be reviewed/completed

destroy an entire bucket brigade. This includes destroying all of the buckets within the bucket brigade's bucket list.

```
$ret = $b->destroy();
```

- **arg1: \$b (APR::Brigade)**

The bucket brigade to destroy

- **ret: \$ret (integer)**

28.3.2 *split*

META: Autogenerated - needs to be reviewed/completed

Split a bucket brigade into two, such that the given bucket is the first in the new bucket brigade. This function is useful when a filter wants to pass only the initial part of a brigade to the next filter.

```
$ret = $b->split($e);
```

- **arg1: \$b (APR::Brigade)**

The brigade to split

- **arg2: \$e (APR::Brigade)**

The first element of the new brigade

- **ret: \$ret (APR::Brigade)**

The new brigade

28.3.3 *concat*

META: Autogenerated - needs to be reviewed/completed

```
$a->concat($b);
```

- **arg1: \$a (APR::Brigade)**
- **arg2: \$b (APR::Brigade)**
- **ret: no return value**

28.3.4 *empty*

META: Autogenerated - needs to be reviewed/completed

```
$ret = $brigade->empty();
```

- **arg1: \$brigade (APR::Brigade)**
- **ret: \$ret (integer)**

28.3.5 *insert_head*

META: Autogenerated - needs to be reviewed/completed

```
$brigade->insert_head($bucket);
```

- **arg1: \$brigade (APR::Brigade)**
- **arg2: \$bucket (APR::Brigade)**
- **ret: no return value**

28.3.6 *insert_tail*

META: Autogenerated - needs to be reviewed/completed

```
$brigade->insert_tail($bucket);
```

- **arg1: \$brigade (APR::Brigade)**
- **arg2: \$bucket (APR::Brigade)**
- **ret: no return value**

28.4 See Also

mod_perl 2.0 documentation.

28.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

28.6 Authors

The mod_perl development team and numerous contributors.

29 APR::Bucket - Perl API for XXX

29.1 Synopsis

```
use APR::Bucket ();
```

META: to be completed

29.2 Description

META: to be completed

29.3 API

APR::Bucket provides the following functions and/or methods:

29.3.1 *eos_create*

META: Autogenerated - needs to be reviewed/completed

Each bucket type foo has two initialization functions: `apr_bucket_foo_make` which sets up some already-allocated memory as a bucket of type foo; and `apr_bucket_foo_create` which allocates memory for the bucket, calls `apr_bucket_make_foo`, and initializes the bucket's list pointers. The `apr_bucket_foo_make` functions are used inside the bucket code to change the type of buckets in place; other code should call `apr_bucket_foo_create`. All the initialization functions change nothing if they fail. *

Create an End of Stream bucket. This indicates that there is no more data coming from down the filter stack. All filters should flush at this point.

```
$ret = $list->eos_create();
```

- **arg1: \$list (APR::BucketAlloc)**

The freelist from which this bucket should be allocated

- **ret: \$ret (APR::Bucket)**

The new bucket, or NULL if allocation failed

29.3.2 *flush_create*

META: Autogenerated - needs to be reviewed/completed

Create a flush bucket. This indicates that filters should flush their data. There is no guarantee that they will flush it, but this is the best we can do.

```
$ret = $list->flush_create();
```

- **arg1: \$list (APR::BucketAlloc)**

The freelist from which this bucket should be allocated

- **ret: \$ret (APR::Bucket)**

The new bucket, or NULL if allocation failed

29.3.3 *insert_after*

META: Autogenerated - needs to be reviewed/completed

```
$a->insert_after($b);
```

- **arg1: \$a (APR::Bucket)**
- **arg2: \$b (APR::Bucket)**
- **ret: no return value**

29.3.4 *insert_before*

META: Autogenerated - needs to be reviewed/completed

```
$a->insert_before($b);
```

- **arg1: \$a (APR::Bucket)**
- **arg2: \$b (APR::Bucket)**
- **ret: no return value**

29.3.5 *is_eos*

META: Autogenerated - needs to be reviewed/completed

```
$ret = $bucket->is_eos();
```

- **arg1: \$bucket (APR::Bucket)**
- **ret: \$ret (integer)**

29.3.6 *is_flush*

META: Autogenerated - needs to be reviewed/completed

```
$ret = $bucket->is_flush();
```

- **arg1: \$bucket (APR::Bucket)**
- **ret: \$ret (integer)**

29.3.7 *remove*

META: Autogenerated - needs to be reviewed/completed

```
$bucket->remove();
```

- **arg1:** `$bucket` (**APR::Bucket**)
- **ret:** no return value

29.4 See Also

mod_perl 2.0 documentation.

29.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

29.6 Authors

The mod_perl development team and numerous contributors.

30 APR::Const - Perl Interface for APR Constants

30.1 Synopsis

```
use APR::Const -compile => qw(constant names ...);
```

30.2 Constants

30.2.1 *:common*

```
use APR::Const -compile => qw(:common);
```

The `:common` group is for XXX constants.

30.2.1.1 **APR::SUCCESS**

30.2.2 *:error*

```
use APR::Const -compile => qw(:error);
```

The `:error` group is for XXX constants.

30.2.2.1 **APR::EABOVEROOT**

30.2.2.2 **APR::EABSOLUTE**

30.2.2.3 **APR::EACCES**

30.2.2.4 **APR::EAGAIN**

30.2.2.5 **APR::EBADDATE**

30.2.2.6 **APR::EBADF**

30.2.2.7 **APR::EBADIP**

30.2.2.8 **APR::EBADMASK**

30.2.2.9 **APR::EBADPATH**

30.2.2.10 **APR::EBUSY**

30.2.2.11 **APR::ECONNABORTED**

30.2.2.12 APR::ECONNREFUSED

30.2.2.13 APR::ECONNRESET

30.2.2.14 APR::EDSOOPEN

30.2.2.15 APR::EEXIST

30.2.2.16 APR::EFTYPE

30.2.2.17 APR::EGENERAL

30.2.2.18 APR::EHOSTUNREACH

30.2.2.19 APR::EINCOMPLETE

30.2.2.20 APR::EINIT

30.2.2.21 APR::EINPROGRESS

30.2.2.22 APR::EINTR

30.2.2.23 APR::EINVAL

30.2.2.24 APR::EINVALSOCK

30.2.2.25 APR::EMFILE

30.2.2.26 APR::EMISMATCH

30.2.2.27 APR::ENAMETOOLONG

30.2.2.28 APR::END

30.2.2.29 APR::ENETUNREACH

30.2.2.30 APR::ENFILE

30.2.2.31 APR::ENODIR

30.2.2.32 APR::ENOENT

30.2.2.33 APR::ENOLOCK

30.2.2.34 APR::ENOMEM

30.2.2.35 APR::ENOPOLL

30.2.2.36 APR::ENOPOOL

30.2.2.37 APR::ENOPROC

30.2.2.38 APR::ENOSHMAVAIL

30.2.2.39 APR::ENOSOCKET

30.2.2.40 APR::ENOSPC

30.2.2.41 APR::ENOSTAT

30.2.2.42 APR::ENOTDIR

30.2.2.43 APR::ENOTEMPTY

30.2.2.44 APR::ENOTHDKEY

30.2.2.45 APR::ENOTHREAD

30.2.2.46 APR::ENOTIME

30.2.2.47 APR::ENOTIMPL

30.2.2.48 APR::ENOTSOCK

30.2.2.49 APR::EOF

30.2.2.50 APR::EPIPE

30.2.2.51 APR::ERELATIVE

30.2.2.52 APR::ESPIPE

30.2.2.53 APR::ETIMEDOUT

30.2.2.54 APR::EXDEV

30.2.3 :filemode

```
use APR::Const -compile => qw(:filemode);
```

The :filemode group is for XXX constants.

30.2.3.1 APR::BINARY

30.2.3.2 APR::BUFFERED

30.2.3.3 APR::CREATE

30.2.3.4 APR::DEONCLOSE

30.2.3.5 APR::EXCL

30.2.3.6 APR::PEND

30.2.3.7 APR::READ

30.2.3.8 APR::TRUNCATE

30.2.3.9 APR::WRITE

30.2.4 :filepath

```
use APR::Const -compile => qw(:filepath);
```

The :filepath group is for XXX constants.

30.2.4.1 APR::FILEPATH_NATIVE

30.2.4.2 APR::FILEPATH_NOTABOVEROOT

30.2.4.3 APR::FILEPATH_NOTABSOLUTE

30.2.4.4 APR::FILEPATH_NOTRELATIVE

30.2.4.5 APR::FILEPATH_SECUREROOT

30.2.4.6 APR::FILEPATH_SECUREROOTTEST

30.2.4.7 APR::FILEPATH_TRUENAME

30.2.5 :fileperms

```
use APR::Const -compile => qw(:fileperms);
```

The :fileperms group is for XXX constants.

30.2.5.1 APR::GEXECUTE

30.2.5.2 APR::GREAD

30.2.5.3 APR::GWRITE

30.2.5.4 APR::UEXECUTE

30.2.5.5 APR::UREAD

30.2.5.6 APR::UWRITE

30.2.5.7 APR::WEEXECUTE

30.2.5.8 APR::WREAD

30.2.5.9 APR::WWRITE

30.2.6 :filetype

```
use APR::Const -compile => qw(:filetype);
```

The :filetype group is for XXX constants.

30.2.6.1 APR::NOFILE

30.2.6.2 APR::REG

30.2.6.3 APR::DIR

30.2.6.4 APR::CHR

30.2.6.5 APR::BLK

30.2.6.6 APR::PIPE

30.2.6.7 APR::LNK

30.2.6.8 APR::SOCK

30.2.6.9 APR::UNKFILE

30.2.7 :*finfo*

```
use APR::Const -compile => qw(:finfo);
```

The :finfo group is for XXX constants.

30.2.7.1 APR::FINFO_ETIME

30.2.7.2 APR::FINFO_CSIZE

30.2.7.3 APR::FINFO_CTIME

30.2.7.4 APR::FINFO_DEV

30.2.7.5 APR::FINFO_DIRENT

30.2.7.6 APR::FINFO_GPROT

30.2.7.7 APR::FINFO_GROUP

30.2.7.8 APR::FINFO_ICASE

30.2.7.9 APR::FINFO_IDENT

30.2.7.10 APR::FINFO_INODE

30.2.7.11 APR::FINFO_LINK

30.2.7.12 APR::FINFO_MIN

30.2.7.13 APR::FINFO_MTIME

30.2.7.14 APR::FINFO_NAME

30.2.7.15 APR::FINFO_NLINK

30.2.7.16 APR::FINFO_NORM

30.2.7.17 APR::FINFO_OWNER

30.2.7.18 APR::FINFO_PROT

30.2.7.19 APR::FINFO_SIZE

30.2.7.20 APR::FINFO_TYPE

30.2.7.21 APR::FINFO_UPROT

30.2.7.22 APR::FINFO_USER

30.2.7.23 APR::FINFO_WPROT

30.2.8 :flock

```
use APR::Const -compile => qw(:flock);
```

The :flock group is for XXX constants.

30.2.8.1 APR::FLOCK_EXCLUSIVE

30.2.8.2 APR::FLOCK_NONBLOCK

30.2.8.3 APR::FLOCK_SHARED

30.2.8.4 APR::FLOCK_TYPEMASK

30.2.9 :hook

```
use APR::Const -compile => qw(:hook);
```

The :hook group is for XXX constants.

30.2.9.1 APR::HOOK_FIRST

30.2.9.2 APR::HOOK_LAST

30.2.9.3 APR::HOOK_MIDDLE

30.2.9.4 APR::HOOK_REALLY_FIRST

30.2.10 :limit

30.2.9.5 APR::HOOK_REALLY_LAST

30.2.10 :limit

```
use APR::Const -compile => qw(:limit);
```

The :limit group is for XXX constants.

30.2.10.1 APR::LIMIT_CPU

30.2.10.2 APR::LIMIT_MEM

30.2.10.3 APR::LIMIT_NOFILE

30.2.10.4 APR::LIMIT_NPROC

30.2.11 :lockmech

```
use APR::Const -compile => qw(:lockmech);
```

The :lockmech group is for XXX constants.

30.2.11.1 APR::LOCK_DEFAULT

30.2.11.2 APR::LOCK_FCNTL

30.2.11.3 APR::LOCK_FLOCK

30.2.11.4 APR::LOCK_POSIXSEM

30.2.11.5 APR::LOCK_PROC_PTHREAD

30.2.11.6 APR::LOCK_SYSVSEM

30.2.12 :poll

```
use APR::Const -compile => qw(:poll);
```

The :poll group is for XXX constants.

30.2.12.1 APR::POLLERR

30.2.12.2 APR::POLLHUP

30.2.12.3 APR::POLLIN**30.2.12.4 APR::POLLNVAL****30.2.12.5 APR::POLLOUT****30.2.12.6 APR::POLLPRI*****30.2.13 :read_type***

```
use APR::Const -compile => qw(:read_type);
```

The :read_type group is for XXX constants.

30.2.13.1 APR::BLOCK_READ**30.2.13.2 APR::NONBLOCK_READ*****30.2.14 :shutdown_how***

```
use APR::Const -compile => qw(:shutdown_how);
```

The :shutdown_how group is for XXX constants.

30.2.14.1 APR::SHUTDOWN_READ**30.2.14.2 APR::SHUTDOWN_READWRITE****30.2.14.3 APR::SHUTDOWN_WRITE*****30.2.15 :socket***

```
use APR::Const -compile => qw(:socket);
```

The :socket group is for XXX constants.

30.2.15.1 APR::SO_DEBUG**30.2.15.2 APR::SO_DISCONNECTED****30.2.15.3 APR::SO_KEEPALIVE****30.2.15.4 APR::SO_LINGER**

30.2.16 :table

30.2.15.5 APR::SO_NONBLOCK

30.2.15.6 APR::SO_RCVBUF

30.2.15.7 APR::SO_REUSEADDR

30.2.15.8 APR::SO_SNDBUF

30.2.16 :table

```
use APR::Const -compile => qw(:table);
```

The :table group is for overlap() and compress() constants. See APR::Table for details.

30.2.16.1 APR::OVERLAP_TABLES_MERGE

30.2.16.2 APR::OVERLAP_TABLES_SET

30.2.17 :uri

```
use APR::Const -compile => qw(:uri);
```

The :uri group is for XXX constants.

30.2.17.1 APR::URI_ACAP_DEFAULT_PORT

30.2.17.2 APR::URI_FTP_DEFAULT_PORT

30.2.17.3 APR::URI_GOPHER_DEFAULT_PORT

30.2.17.4 APR::URI_HTTPS_DEFAULT_PORT

30.2.17.5 APR::URI_HTTP_DEFAULT_PORT

30.2.17.6 APR::URI_IMAP_DEFAULT_PORT

30.2.17.7 APR::URI_LDAP_DEFAULT_PORT

30.2.17.8 APR::URI_NFS_DEFAULT_PORT

30.2.17.9 APR::URI_NNTP_DEFAULT_PORT

30.2.17.10 APR::URI_POP_DEFAULT_PORT

30.2.17.11 `APR::URI_PROSPERO_DEFAULT_PORT`

30.2.17.12 `APR::URI_RTSP_DEFAULT_PORT`

30.2.17.13 `APR::URI_SIP_DEFAULT_PORT`

30.2.17.14 `APR::URI_SNEWS_DEFAULT_PORT`

30.2.17.15 `APR::URI_SSH_DEFAULT_PORT`

30.2.17.16 `APR::URI_TELNET_DEFAULT_PORT`

30.2.17.17 `APR::URI_TIP_DEFAULT_PORT`

30.2.17.18 `APR::URI_UNP_OMITPASSWORD`

30.2.17.19 `APR::URI_UNP_OMITPATHINFO`

30.2.17.20 `APR::URI_UNP_OMITQUERY`

30.2.17.21 `APR::URI_UNP_OMITSITEPART`

30.2.17.22 `APR::URI_UNP_OMITUSER`

30.2.17.23 `APR::URI_UNP_OMITUSERINFO`

30.2.17.24 `APR::URI_UNP_REVEALPASSWORD`

30.2.17.25 `APR::URI_WAIS_DEFAULT_PORT`

30.3 See Also

`mod_perl 2.0` documentation.

30.4 Copyright

`mod_perl 2.0` and its core modules are copyrighted under The Apache Software License, Version 1.1.

30.5 Authors

The `mod_perl` development team and numerous contributors.

31 APR::Date - Perl API for XXX

31.1 Synopsis

```
use APR::Date ();
```

META: to be completed

31.2 Description

META: to be completed

31.3 API

APR::Date provides the following functions and/or methods:

31.3.1 *parse_http*

META: Autogenerated - needs to be reviewed/completed

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
```

```
$ret = parse_http($date);
```

- **arg1: \$date (string)**

The date in one of the three formats above

- **ret: \$ret (number)**

the `apr_time_t` number of microseconds since 1 Jan 1970 GMT, or 0 if this would be out of range or if the date is invalid.

31.3.2 *parse_rfc*

META: Autogenerated - needs to be reviewed/completed

Parses a string resembling an RFC 822 date. This is meant to be leinent in its parsing of dates. Hence, this will parse a wider range of dates than `apr_date_parse_http`.

The prominent mailer (or poster, if mailer is unknown) that has been seen in the wild is included for the unknown formats:

31.4 See Also

```
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
Sun, 6 Nov 1994 08:49:37 GMT  ; RFC 822, updated by RFC 1123
Sun, 06 Nov 94 08:49:37 GMT   ; RFC 822
Sun, 6 Nov 94 08:49:37 GMT    ; RFC 822
Sun, 06 Nov 94 08:49 GMT      ; Unknown [drtr\@ast.cam.ac.uk]
Sun, 6 Nov 94 08:49 GMT       ; Unknown [drtr\@ast.cam.ac.uk]
Sun, 06 Nov 94 8:49:37 GMT    ; Unknown [Elm 70.85]
Sun, 6 Nov 94 8:49:37 GMT     ; Unknown [Elm 70.85]
```

```
$ret = parse_rfc($date);
```

- **arg1: \$date (string)**

The date in one of the formats above

- **ret: \$ret (number)**

the `apr_time_t` number of microseconds since 1 Jan 1970 GMT, or 0 if this would be out of range or if the date is invalid.

31.4 See Also

`mod_perl 2.0 documentation`.

31.5 Copyright

`mod_perl 2.0` and its core modules are copyrighted under The Apache Software License, Version 1.1.

31.6 Authors

The `mod_perl` development team and numerous contributors.

32 APR::Finfo - Perl API for XXX

32.1 Synopsis

```
use APR::Finfo ();
```

META: to be completed

32.2 Description

META: to be completed

32.3 API

APR::Finfo provides the following functions and/or methods:

32.3.1 *stat*

META: Autogenerated - needs to be reviewed/completed

get the specified file's stats. The file is specified by filename, instead of using a pre-opened file.

```
$ret = $finfo->stat($fname, $wanted, $cont);
```

- **arg1: \$finfo (APR::Finfo)**

Where to store the information about the file, which is never touched if the call fails.

- **arg2: \$fname (string)**

The name of the file to stat.

- **arg3: \$wanted (string)**

The desired apr_finfo_t fields, as a bit flag of APR_FINFO_ values

- **arg4: \$cont (integer)**

the pool to use to allocate the new file.

- **ret: \$ret (integer)**

32.3.2 *pool*

META: Autogenerated - needs to be reviewed/completed

Allocates memory and closes lingering handles in the specified pool


```
$ret = $obj->pool($newval);
```

- **arg1:** \$obj (APR::Finfo)
- **arg2:** \$newval (APR::Pool)

32.3.3 *valid*

META: Autogenerated - needs to be reviewed/completed

The bitmask describing valid fields of this apr_finfo_t structure including all available 'wanted' fields and potentially more

```
$ret = $obj->valid($newval);
```

- **arg1:** \$obj (APR::Finfo)
- **arg2:** \$newval (integer)

32.3.4 *protection*

META: Autogenerated - needs to be reviewed/completed

The access permissions of the file. Mimics Unix access rights.

```
$ret = $obj->protection($newval);
```

- **arg1:** \$obj (APR::Finfo)
- **arg2:** \$newval (integer)

32.3.5 *filetype*

META: Autogenerated - needs to be reviewed/completed

The type of file. One of APR_REG, APR_DIR, APR_CHR, APR_BLK, APR_PIPE, APR_LNK or APR SOCK. If the type is undetermined, the value is APR_NOFILE. If the type cannot be determined, the value is APR_UNKFILE.

```
$ret = $obj->filetype($newval);
```

- **arg1:** \$obj (APR::Finfo)
- **arg2:** \$newval (integer)

32.3.6 *user*

META: Autogenerated - needs to be reviewed/completed

32.3.7 group

The user id that owns the file

```
$ret = $obj->user($newval);
```

- **arg1: \$obj (APR::Finfo)**
- **arg2: \$newval (integer)**

32.3.7 *group*

META: Autogenerated - needs to be reviewed/completed

The group id that owns the file

```
$ret = $obj->group($newval);
```

- **arg1: \$obj (APR::Finfo)**
- **arg2: \$newval (integer)**

32.3.8 *inode*

META: Autogenerated - needs to be reviewed/completed

The inode of the file.

```
$ret = $obj->inode($newval);
```

- **arg1: \$obj (APR::Finfo)**
- **arg2: \$newval (integer)**

32.3.9 *device*

META: Autogenerated - needs to be reviewed/completed

The id of the device the file is on.

```
$ret = $obj->device($newval);
```

- **arg1: \$obj (APR::Finfo)**
- **arg2: \$newval (number)**

32.3.10 *nlink*

META: Autogenerated - needs to be reviewed/completed

The number of hard links to the file.

```
$ret = $obj->nlink($newval);
```

- **arg1: \$obj (APR::Finfo)**
- **arg2: \$newval (integer)**

32.3.11 size

META: Autogenerated - needs to be reviewed/completed

The size of the file

```
$ret = $obj->size($newval);
```

- **arg1: \$obj (APR::Finfo)**
- **arg2: \$newval (integer)**

32.3.12 csize

META: Autogenerated - needs to be reviewed/completed

The storage size consumed by the file

```
$ret = $obj->csize($newval);
```

- **arg1: \$obj (APR::Finfo)**
- **arg2: \$newval (integer)**

32.3.13 atime

META: Autogenerated - needs to be reviewed/completed

The time the file was last accessed

```
$ret = $obj->atime($newval);
```

- **arg1: \$obj (APR::Finfo)**
- **arg2: \$newval (number)**

32.3.14 mtime

META: Autogenerated - needs to be reviewed/completed

The time the file was last modified

```
$ret = $obj->mtime($newval);
```

- **arg1: \$obj (APR::Finfo)**
- **arg2: \$newval (number)**

32.3.15 ctime

META: Autogenerated - needs to be reviewed/completed

The time the file was last changed

```
$ret = $obj->ctime($newval);
```

- **arg1: \$obj (APR::Finfo)**
- **arg2: \$newval (number)**

32.3.16 fname

META: Autogenerated - needs to be reviewed/completed

The pathname of the file (possibly unrooted)

```
$ret = $obj->fname($newval);
```

- **arg1: \$obj (APR::Finfo)**
- **arg2: \$newval (string)**

32.3.17 name

META: Autogenerated - needs to be reviewed/completed

The file's name (no path) in filesystem case

```
$ret = $obj->name($newval);
```

- **arg1: \$obj (APR::Finfo)**
- **arg2: \$newval (string)**

32.4 See Also

mod_perl 2.0 documentation.

32.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

32.6 Authors

The mod_perl development team and numerous contributors.

33 APR::NetLib - Perl API for XXX

33.1 Synopsis

```
use APR::NetLib ();
```

META: to be completed

33.2 Description

META: to be completed

33.3 API

APR::NetLib provides the following functions and/or methods:

33.3.1 *test*

META: Autogenerated - needs to be reviewed/completed

Test the IP address in an `apr_sockaddr_t` against a pre-built ip-subnet representation.

```
$ret = $ipsub->test($sa);
```

- **arg1: \$ipsub (APR::IpSubnet)**

The ip-subnet representation

- **arg2: \$sa (APR::SockAddr)**

The socket address to test

- **ret: \$ret (integer)**

non-zero if the socket address is within the subnet, 0 otherwise

33.4 See Also

mod_perl 2.0 documentation.

33.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

33.6 Authors

The mod_perl development team and numerous contributors.

34 APR::PerlIO -- An APR Perl IO layer

34.1 Synopsis

```
# under mod_perl
use APR::Perlio ();

sub handler {
    my $r = shift;

    die "This Perl build doesn't support Perlio layers"
        unless APR::Perlio::PERLIO_LAYERS_ARE_ENABLED;

    open my $fh, ">:APR", $filename, $r->pool or die $!;
    # work with $fh as normal $fh
    close $fh;

    return Apache::OK;
}

# outside mod_perl
% perl -MApache2 -MAPR -MAPR::Perlio -MAPR::Pool -le \
'open my $fh, ">:APR", "/tmp/apr", APR::Pool->new or die "$!"; \
print $fh "whoah!"; \
close $fh;'
```

34.2 Description

APR::Perlio implements a Perl IO layer using APR's file manipulation as its internals.

Why do you want to use this? Normally you shouldn't, probably it won't be faster than Perl's default layer. It's only useful when you need to manipulate a filehandle opened at the APR side, while using Perl.

Normally you won't call open() with APR layer attribute, but some mod_perl functions will return a filehandle which is internally hooked to APR. But you can use APR Perl IO directly if you want.

34.3 Constants

34.3.1 *APR::Perlio::PERLIO_LAYERS_ARE_ENABLED*

Before using the Perl IO APR layer one has to check whether it's supported by the used perl build.

```
die "This Perl build doesn't support Perlio layers"
    unless APR::Perlio::PERLIO_LAYERS_ARE_ENABLED;
```

Notice that loading APR::Perlio won't fail when Perl IO layers aren't available since APR::Perlio provides functionality for Perl builds not supporting Perl IO layers.

34.4 API

Perl Interface:

34.4.1 *open*

Open a file via APR Perl IO layer.

```
open my $fh, ">:APR", $filename, $r->pool or die $!;
```

- **arg1: \$fh (GLOB filehandle)**

The filehandle.

- **arg2: \$mode (string)**

The mode to open the file, constructed from two sections separated by the `:` character: the first section is the mode to open the file under (`>`, `<`, etc) and the second section must be a string *APR*. For more information refer to the *open* entry in the *perlfunc* manpage.

- **arg3: \$filename (string)**

The path to the filename to open

- **arg4: \$p (APR::Pool)**

The pool object to use to allocate APR::PerlIO layer.

- **ret: success or failure**

34.4.2 *seek()*

Sets `$fh`'s position, just like the `seek()` perl call:

```
seek($fh, $offset, $whence);
```

If `$offset` is zero, `seek()` works normally.

However if `$offset` is non-zero and Perl has been compiled with large files support (`-Duse-largefiles`), whereas APR wasn't, this function will croak. This is because largefile size `Off_t` simply cannot fit into a non-largefile size `apr_off_t`.

To solve the problem, rebuild Perl with `-Uuselargefiles`. Currently there is no way to force APR to build with large files support.

34.5 C API

The C API provides functions to convert between Perl IO and APR Perl IO filehandles.

META: document these

34.6 See Also

mod_perl 2.0 documentation. The *perliol(1)*, *perlpio(1)* and *perl(1)* manpages.

34.7 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

34.8 Authors

The mod_perl development team and numerous contributors.

35 APR::Pool - Perl API for XXX

35.1 Synopsis

```
use APR::Pool ();
```

META: to be completed

35.2 Description

META: to be completed

35.3 API

APR::Pool provides the following functions and/or methods:

35.3.1 *cleanup_for_exec*

META: Autogenerated - needs to be reviewed/completed

buffers, *don't* wait for subprocesses, and *don't* free any memory. * Run all of the child_cleanups, so that any unnecessary files are closed because we are about to exec a new program

- **ret: no return value**

35.3.2 *clear*

META: Autogenerated - needs to be reviewed/completed

Clear all memory in the pool and run all the cleanups. This also destroys all subpools.

```
$p->clear();
```

- **arg1: \$p (APR::Pool)**

The pool to clear

- **ret: no return value**

This does not actually free the memory, it just allows the pool to re-use this memory for the next allocation.

35.3.3 *destroy*

META: Autogenerated - needs to be reviewed/completed

Destroy the pool. This takes similar action as `apr_pool_clear()` and then frees all the memory.

```
$p->destroy();
```

- **arg1: \$p (APR::Pool)**

The pool to destroy

- **ret: no return value**

This will actually free the memory

35.3.4 *is_ancestor*

META: Autogenerated - needs to be reviewed/completed

Determine if pool a is an ancestor of pool b

```
$ret = $a->is_ancestor($b);
```

- **arg1: \$a (APR::Pool)**

The pool to search

- **arg2: \$b (APR::Pool)**

The pool to search for

- **ret: \$ret (integer)**

True if a is an ancestor of b, NULL is considered an ancestor of all pools.

35.3.5 *tag*

META: Autogenerated - needs to be reviewed/completed

Tag a pool (give it a name)

```
$pool->tag($tag);
```

- **arg1: \$pool (APR::Pool)**

The pool to tag

- **arg2: \$tag (string)**

The tag

- **ret:** no return value

35.4 See Also

`mod_perl` 2.0 documentation.

35.5 Copyright

`mod_perl` 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

35.6 Authors

The `mod_perl` development team and numerous contributors.

36 APR::SockAddr - Perl API for XXX

36.1 Synopsis

```
use APR::SockAddr ();
```

META: to be completed

36.2 Description

META: to be completed

36.3 API

APR::SockAddr provides the following functions and/or methods:

36.3.1 *equal*

META: Autogenerated - needs to be reviewed/completed

See if the IP addresses in two APR socket addresses are equivalent. Appropriate logic is present for comparing IPv4-mapped IPv6 addresses with IPv4 addresses.

```
$ret = $addr1->equal($addr2);
```

- **arg1: \$addr1 (APR::SockAddr)**

One of the APR socket addresses.

- **arg2: \$addr2 (APR::SockAddr)**

The other APR socket address.

- **ret: \$ret (integer)**

The return value will be non-zero if the addresses are equivalent.

36.4 See Also

mod_perl 2.0 documentation.

36.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

36.6 Authors

The mod_perl development team and numerous contributors.

37 APR::Socket - Perl API for XXX

37.1 Synopsis

```
use APR::Socket ();
```

META: to be completed

37.2 Description

META: to be completed

37.3 API

APR::Socket provides the following functions and/or methods:

37.3.1 *bind*

META: Autogenerated - needs to be reviewed/completed

Bind the socket to its associated port

```
$ret = $sock->bind($sa);
```

- **arg1: \$sock (APR::Socket)**

The socket to bind

- **arg2: \$sa (APR::SockAddr)**

The socket address to bind to

- **ret: \$ret (integer)**

This may be where we will find out if there is any other process using the selected port.

37.3.2 *close*

META: Autogenerated - needs to be reviewed/completed

Close a socket.

```
$ret = $thesocket->close();
```

- **arg1: \$thesocket (APR::Socket)**

The socket to close

- **ret: \$ret (integer)**

37.3.3 connect

META: Autogenerated - needs to be reviewed/completed

Issue a connection request to a socket either on the same machine or a different one.

```
$ret = $sock->connect($sa);
```

- **arg1: \$sock (APR::Socket)**

The socket we wish to use for our side of the connection

- **arg2: \$sa (APR::SockAddr)**

The address of the machine we wish to connect to. If NULL, APR assumes that the sockaddr_in in the apr_socket is completely filled out.

- **ret: \$ret (integer)**

37.3.4 listen

META: Autogenerated - needs to be reviewed/completed

Listen to a bound socket for connections.

```
$ret = $sock->listen($backlog);
```

- **arg1: \$sock (APR::Socket)**

The socket to listen on

- **arg2: \$backlog (integer)**

The number of outstanding connections allowed in the sockets listen queue. If this value is less than zero, the listen queue size is set to zero.

- **ret: \$ret (integer)**

37.3.5 opt_get

META: Autogenerated - needs to be reviewed/completed

Query socket options for the specified socket

```
$ret = $sock->opt_get($opt, $on);
```

- **arg1: \$sock (APR::Socket)**

The socket to query

- **arg2: \$opt (integer)**

The option we would like to query. One of:

```
APR::SO_DEBUG          -- turn on debugging information
APR::SO_KEEPALIVE      -- keep connections active
APR::SO_LINGER         -- lingers on close if data is present
APR::SO_NONBLOCK       -- Turns blocking on/off for socket
APR::SO_REUSEADDR      -- The rules used in validating addresses
                        supplied to bind should allow reuse
                        of local addresses.
APR::SO_SNDBUF         -- Set the SendBufferSize
APR::SO_RCVBUF         -- Set the ReceiveBufferSize
APR::SO_DISCONNECTED -- Query the disconnected state of the socket.
                        (Currently only used on Windows)
```

- **arg3: \$on (integer)**

Socket option returned on the call.

- **ret: \$ret (integer)**

37.3.6 opt_set

META: Autogenerated - needs to be reviewed/completed

Setup socket options for the specified socket

```
$ret = $sock->opt_set($opt, $on);
```

- **arg1: \$sock (APR::Socket)**

The socket to set up.

- **arg2: \$opt (integer)**

The option we would like to configure. One of:

```
APR::SO_DEBUG          -- turn on debugging information
APR::SO_KEEPALIVE      -- keep connections active
APR::SO_LINGER         -- lingers on close if data is present
APR::SO_NONBLOCK       -- Turns blocking on/off for socket
APR::SO_REUSEADDR      -- The rules used in validating addresses
                        supplied to bind should allow reuse of local
                        addresses.
APR::SO_SNDBUF         -- Set the SendBufferSize
APR::SO_RCVBUF         -- Set the ReceiveBufferSize
```

- **arg3: \$on (integer)**

Value for the option.

- **ret: \$ret (integer)**

37.3.7 *recvfrom*

META: Autogenerated - needs to be reviewed/completed

```
$ret = $from->recvfrom($sock, $flags, $buf, $len);
```

- **arg1: \$from (APR::SockAddr)**

The apr_sockaddr_t to fill in the recipient info

- **arg2: \$sock (APR::SockAddr)**

The socket to use

- **arg3: \$flags (integer)**

The flags to use

- **arg4: \$buf (integer)**

The buffer to use

- **arg5: \$len (string)**

The length of the available buffer

- **ret: \$ret (integer)**

37.3.8 *sendto*

META: Autogenerated - needs to be reviewed/completed

```
$ret = $sock->sendto($where, $flags, $buf, $len);
```

- **arg1: \$sock (APR::Socket)**

The socket to send from

- **arg2: \$where (APR::Socket)**

The apr_sockaddr_t describing where to send the data

- **arg3: \$flags (integer)**

The flags to use

- **arg4: \$buf (scalar)**

The data to send

- **arg5: \$len (string)**

The length of the data to send

- **ret: \$ret (integer)**

37.3.9 *timeout_set*

META: Autogenerated - needs to be reviewed/completed

Setup socket timeout for the specified socket

```
$ret = $sock->timeout_set($t);
```

- **arg1: \$sock (APR::Socket)**

The socket to set up.

- **arg2: \$t (number)**

Value for the timeout:

```
t > 0  -- read and write calls return APR::TIMEUP if specified time
        elapses with no data read or written
t == 0  -- read and write calls never block
t < 0  -- read and write calls block
```

- **ret: \$ret (integer)**

37.4 See Also

mod_perl 2.0 documentation.

37.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

37.6 Authors

The mod_perl development team and numerous contributors.

38 APR::Table - Perl API for for manipulating opaque string-content table

38.1 Synopsis

```

use APR::Table ();

$table = make($pool, $nelts);
$table_copy = $table->copy($pool);

$table->clear();

$table->set($key => $val);
$table->unset($key);
$table->add($key, $val);

$val = $table->get($key);
@val = $table->get($key);

$table->merge($key => $val);
overlap($table_a, $table_b, $flags);
$new_table = overlay($table_base, $table_overlay, $pool);

$table->do(sub {print "key $_[0], value $_[1]\n"}, @valid_keys);

#Tied Interface
$value = $table->{$key};
$table->{$key} = $value;
$table->{$key} = $value;
exists $table->{$key};

foreach my $key (keys %{$table}) {
    print "$key = $table->{$key}\n";
}

```

38.2 Description

APR::Table allows its users to manipulate opaque string-content tables.

The table's structure is somewhat similar to the Perl's hash structure, but allows multiple values for the same key. An access to the records stored in the table always requires a key.

The key-value pairs are stored in the order they are added.

The keys are case-insensitive.

However as of the current implementation if more than value for the same key is requested, the whole table is lineary searched, which is very inefficient unless the table is very small.

APR::Table provides a TIE Interface.

See *apr/include/apr_tables.h* in ASF's *apr* project for low level details.

38.3 API

APR::Table provides the following functions and/or methods:

38.3.1 *add*

Add data to a table, regardless of whether there is another element with the same key.

```
$t->add($key, $val);
```

- **arg1: \$t (APR::Table)**

The table to add to.

- **arg2: \$key (string)**

The key to use.

- **arg3: \$val (string)**

The value to add.

- **ret: no return value**

When adding data, this function makes a copy of both the key and the value.

38.3.2 *clear*

Delete all of the elements from a table.

```
$t->clear();
```

- **arg1: \$t (APR::Table)**

The table to clear.

- **ret: no return value**

38.3.3 *compress*

Eliminate redundant entries in a table by either overwriting or merging duplicates:

```
$t->compress($flags);
```

- **arg1: \$t (APR::Table)**

The table to compress.

- **arg2: \$flags (integer)**

```
APR::OVERLAP_TABLES_MERGE -- to merge
APR::OVERLAP_TABLES_SET   -- to overwrite
```

- **ret: no return value**

Converts multi-valued keys in `$table` to single-valued keys. This function takes duplicate table entries and flattens them into a single entry. The flattening behavior is controlled by the (mandatory) `$flags` argument.

When `$flags == APR::OVERLAP_TABLES_SET`, each key will be set to the last value seen for that key. For example, given key/value pairs 'foo => bar' and 'foo => baz', 'foo' would have a final value of 'baz' after compression - the 'bar' value would be lost.

When `$flags == APR::OVERLAP_TABLES_MERGE`, multiple values for the same key are flattened into a comma-separated list. Given key/value pairs 'foo => bar' and 'foo => baz', 'foo' would have a final value of 'bar, baz' after compression.

38.3.4 copy

Create a new table and copy another table into it.

```
$ret = $t->copy($p);
```

- **arg1: \$t (APR::Table)**

The table to copy.

- **arg2: \$p (APR::Pool)**

The pool to allocate the new table out of.

- **ret: \$ret (APR::Table)**

A copy of the table passed in.

38.3.5 do

Iterate over all the elements of the table, invoking provided subroutine for each element. The subroutine gets passed as argument, a key-value pair.

```
$table->do(sub {...}, @filter);
```

- **arg1: \$p (APR::Table)**

The table to operate on.

- **arg2: \$sub (CODE ref/string)**

A subroutine reference or name to be called on each item in the table. The subroutine can abort the iteration by returning 0 and should always return 1 otherwise.

- **opt arg3: @filter (ARRAY)**

If passed, only keys matching one of the entries in the @filter will be processed.

- **ret: no return value**

38.3.6 *get*

Get the value(s) associated with a given key. After this call, the data is still in the table.

```
$val = $table->get($key);
@val = $table->get($key);
```

- **arg1: \$t (APR::Table)**

The table to search for the key.

- **arg2: \$key (string)**

The key to search for.

- **ret: \$val or @val**

In the scalar context the first matching value returned. (The oldest in the table, if there is more than one value.)

In the list context the whole table is traversed and all matching values are returned. If nothing matches undef is returned.

38.3.7 *make*

Make a new table.

```
$ret = make($p, $nelts);
```

- **arg1: \$p (APR::Pool)**

The pool to allocate the pool out of.

- **arg2: \$nelts (integer)**

The number of elements in the initial table.

- **ret: \$ret (APR::Table)**

The new table.

Note: This table can only store text data.

38.3.8 *merge*

Add data to a table by merging the value with data that has already been stored:

```
$t->merge($key, $val);
```

- **arg1: \$t (APR::Table)**

The table to search for the data.

- **arg2: \$key (string)**

The key to merge data for.

- **arg3: \$val (string)**

The data to add.

- **ret: no return value**

Note: if the key is not found, then this function acts like `add()`.

38.3.9 *overlap*

For each key/value pair in `$t_b`, add the data to `$t_a`. The definition of `$flags` explains how `$flags` define the overlapping method.

```
$t_a->overlap($t_b, $flags);
```

- **arg1: \$t_a (APR::Table)**

The table to add the data to.

- **arg2: \$t_b (APR::Table)**

The table to iterate over, adding its data to table `$t_a`

- **arg3: \$flags (integer)**

How to add the table to table `$t_a`.

When `$flags == APR::OVERLAP_TABLES_SET`, if another element already exists with the same key, this will over-write the old data.

When `$flags == APR::OVERLAP_TABLES_MERGE`, the key/value pair from `$t_b` is added, regardless of whether there is another element with the same key in `$t_a`.

- **ret: no return value**

This function is highly optimized, and uses less memory and CPU cycles than a function that just loops through table `$t_b` calling other functions.

Conceptually, `overlap()` does this:

```
apr_array_header_t *barr = apr_table_elts(b);
apr_table_entry_t *belt = (apr_table_entry_t *)barr->elts;
int i;

for (i = 0; i < barr->nelts; ++i) {
    if (flags & APR_OVERLAP_TABLES_MERGE) {
        apr_table_mergen(a, belt[i].key, belt[i].val);
    }
    else {
        apr_table_setn(a, belt[i].key, belt[i].val);
    }
}
```

Except that it is more efficient (less space and cpu-time) especially when `$t_b` has many elements.

Notice the assumptions on the keys and values in `$t_b` -- they must be in an ancestor of `$t_a`'s pool. In practice `$t_b` and `$t_a` are usually from the same pool.

38.3.10 *overlay*

Merge two tables into one new table. The resulting table may have more than one value for the same key.

```
$t = $t_base->overlay($t_overlay, $p);
```

- **arg1: \$t_base (APR::Table)**

The table to add at the end of the new table.

- **arg2: \$t_overlay (APR::Table)**

The first table to put in the new table.

- **arg3: \$p (APR::Pool)**

The pool to use for the new table.

- **ret: \$t (APR::Table)**

A new table containing all of the data from the two passed in.

38.3.11 *set*

Add a key/value pair to a table, if another element already exists with the same key, this will over-write the old data.

```
$t->set($key, $val);
```

- **arg1: \$t (APR::Table)**

The table to add the data to.

- **arg2: \$key (string)**

The key to use.

- **arg3: \$val (string)**

The value to add.

- **ret: no return value**

When adding data, this function makes a copy of both the key and the value.

38.3.12 *unset*

Remove data from the table.

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

- **arg1: \$t (APR::Table)**

The table to remove data from.

- **arg2: \$key (string)**

The key of the data being removed.

- **ret: no return value**

38.4 TIE Interface

APR::Table also implements a tied interface, so you can work with the \$table object as a hash reference.

The following tied-hash function are supported: `FETCH`, `STORE`, `DELETE`, `CLEAR`, `EXISTS`, `FIRSTKEY`, `NEXTKEY` and `DESTROY`.

remark: `APR::Table` can hold more than one key-value pair sharing the same key, so when using a table through the tied interface, the first entry found with the right key will be used, completely disregarding possible other entries with the same key. The only exception to this is if you iterate over the list with *each*, then you can access all key-value pairs that share the same key.

38.4.1 *EXISTS*

```
$ret = $t->EXISTS($key);
```

- **arg1:** `$t` (`APR::Table`)
- **arg2:** `$key` (string)
- **ret:** `$ret` (integer)

38.4.2 *CLEAR*

```
$t->CLEAR();
```

- **arg1:** `$t` (`APR::Table`)
- **ret:** no return value

38.4.3 *STORE*

```
$t->STORE($key, $value);
```

- **arg1:** `$t` (`APR::Table`)
- **arg2:** `$key` (string)
- **arg3:** `$value` (string)
- **ret:** no return value

38.4.4 *DELETE*

```
$t->DELETE($key);
```

- **arg1:** `$t` (`APR::Table`)
- **arg2:** `$key` (string)
- **ret:** no return value

38.4.5 *FETCH*

```
$ret = $t->FETCH($key);
```

- **arg1:** `$t` (`APR::Table`)
- **arg2:** `$key` (string)
- **ret:** `$ret` (string)

38.5 See Also

`mod_perl` 2.0 documentation.

38.6 Copyright

`mod_perl` 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

38.7 Authors

The `mod_perl` development team and numerous contributors.

39 APR::ThreadMutex - Perl API for XXX

39.1 Synopsis

```
use APR::ThreadMutex ();
```

META: to be completed

39.2 Description

META: to be completed

39.3 API

APR::ThreadMutex provides the following functions and/or methods:

39.3.1 *DESTROY*

META: Autogenerated - needs to be reviewed/completed

Destroy the mutex and free the memory associated with the lock.

```
$mutex->DESTROY();
```

- **arg1: \$mutex (APR::ThreadMutex)**

the mutex to destroy.

- **ret: no return value**

39.3.2 *lock*

META: Autogenerated - needs to be reviewed/completed

Acquire the lock for the given mutex. If the mutex is already locked, the current thread will be put to sleep until the lock becomes available.

```
$ret = $mutex->lock();
```

- **arg1: \$mutex (APR::ThreadMutex)**

the mutex on which to acquire the lock.

- **ret: \$ret (integer)**

39.3.3 *pool_get*

META: Autogenerated - needs to be reviewed/completed

Get the pool used by this thread_mutex.

```
$ret = $obj->pool_get();
```

- **arg1: \$obj (APR::ThreadMutex)**
- **ret: \$ret (APR::Pool)**

apr_pool_t the pool

39.3.4 *trylock*

META: Autogenerated - needs to be reviewed/completed

Attempt to acquire the lock for the given mutex. If the mutex has already been acquired, the call returns immediately with APR_EBUSY. Note: it is important that the APR_STATUS_IS_EBUSY(s) macro be used to determine if the return value was APR_EBUSY, for portability reasons.

```
$ret = $mutex->trylock();
```

- **arg1: \$mutex (APR::ThreadMutex)**

the mutex on which to attempt the lock acquiring.

- **ret: \$ret (integer)**

39.3.5 *unlock*

META: Autogenerated - needs to be reviewed/completed

Release the lock for the given mutex.

```
$ret = $mutex->unlock();
```

- **arg1: \$mutex (APR::ThreadMutex)**

the mutex from which to release the lock.

- **ret: \$ret (integer)**

39.4 See Also

mod_perl 2.0 documentation.

39.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

39.6 Authors

The mod_perl development team and numerous contributors.

40 APR::URI - Perl API for XXX

40.1 Synopsis

```
use APR::URI ( );
```

META: to be completed

40.2 Description

META: to be completed

40.3 API

APR::URI provides the following functions and/or methods:

40.3.1 *port_of_scheme*

META: Autogenerated - needs to be reviewed/completed

Return the default port for a given scheme. The schemes recognized are http, ftp, https, gopher, wais, nntp, snews, and prospero

```
$ret = port_of_scheme($scheme_str);
```

- **arg1: \$scheme_str (string)**

The string that contains the current scheme

- **ret: \$ret (integer)**

The default port for this scheme

40.3.2 *scheme*

META: Autogenerated - needs to be reviewed/completed

scheme ("http"/"ftp"/...)

```
$ret = $obj->scheme($newval);
```

- **arg1: \$obj (APR::URI)**
- **arg2: \$newval (string)**

40.3.3 hostinfo

META: Autogenerated - needs to be reviewed/completed

combined [user[:password]\@]host[:port]

```
$ret = $obj->hostinfo($newval);
```

- **arg1: \$obj (APR::URI)**
- **arg2: \$newval (string)**

40.3.4 user

META: Autogenerated - needs to be reviewed/completed

user name, as in http://user:passwd\@host:port/

```
$ret = $obj->user($newval);
```

- **arg1: \$obj (APR::URI)**
- **arg2: \$newval (string)**

40.3.5 password

META: Autogenerated - needs to be reviewed/completed

password, as in http://user:passwd\@host:port/

```
$ret = $obj->password($newval);
```

- **arg1: \$obj (APR::URI)**
- **arg2: \$newval (string)**

40.3.6 hostname

META: Autogenerated - needs to be reviewed/completed

hostname from URI (or from Host: header)

```
$ret = $obj->hostname($newval);
```

- **arg1: \$obj (APR::URI)**
- **arg2: \$newval (string)**

40.3.7 *path*

META: Autogenerated - needs to be reviewed/completed

the request path (or "/" if only scheme://host was given)

```
$ret = $obj->path($newval);
```

- **arg1: \$obj (APR: :URI)**
- **arg2: \$newval (string)**

40.3.8 *query*

META: Autogenerated - needs to be reviewed/completed

Everything after a '?' in the path, if present

```
$ret = $obj->query($newval);
```

- **arg1: \$obj (APR: :URI)**
- **arg2: \$newval (string)**

40.3.9 *fragment*

META: Autogenerated - needs to be reviewed/completed

Trailing "#fragment" string, if present

```
$ret = $obj->fragment($newval);
```

- **arg1: \$obj (APR: :URI)**
- **arg2: \$newval (string)**

40.3.10 *is_initialized*

META: Autogenerated - needs to be reviewed/completed

has the structure been initialized

```
$ret = $obj->is_initialized($newval);
```

- **arg1: \$obj (APR: :URI)**
- **arg2: \$newval (number)**

40.3.11 dns_looked_up

META: Autogenerated - needs to be reviewed/completed

has the DNS been looked up yet

```
$ret = $obj->dns_looked_up($newval);
```

- **arg1: \$obj (APR::URI)**
- **arg2: \$newval (number)**

40.3.12 dns_resolved

META: Autogenerated - needs to be reviewed/completed

has the dns been resolved yet

```
$ret = $obj->dns_resolved($newval);
```

- **arg1: \$obj (APR::URI)**
- **arg2: \$newval (number)**

40.4 See Also

mod_perl 2.0 documentation.

40.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

40.6 Authors

The mod_perl development team and numerous contributors.

41 APR::Util - Perl API for XXX

41.1 Synopsis

```
use APR::Util ();
```

META: to be completed

41.2 Description

META: to be completed

41.3 API

APR::Util provides the following functions and/or methods:

41.3.1 filepath_name_get

META: Autogenerated - needs to be reviewed/completed

return the final element of the pathname

```
$ret = filepath_name_get($pathname);
```

- **arg1: \$pathname (string)**

The path to get the final element of

- **ret: \$ret (string)**

the final element of the path @remark <PRE> For example: "/foo/bar/gum" -> "gum"
"/foo/bar/gum/" -> "" "gum" -> "gum" "bs\\path\\stuff" -> "stuff" </PRE>

41.3.2 password_get

META: Autogenerated - needs to be reviewed/completed

Display a prompt and read in the password from stdin.

```
$ret = password_get($prompt, $pwbuf, $bufsize);
```

- **arg1: \$prompt (string)**

The prompt to display

- **arg2: \$pwbuf (string)**

Buffer to store the password

41.4 See Also

- **arg3: \$bufsize (number)**

The length of the password buffer.

- **ret: \$ret (integer)**

41.4 See Also

mod_perl 2.0 documentation.

41.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

41.6 Authors

The mod_perl development team and numerous contributors.

42 ModPerl::MethodLookup -- Map mod_perl 2.0 modules, objects and methods

42.1 Synopsis

```

use ModPerl::MethodLookup;

# return all module names containing XS method 'print'
my($hint, @modules) =
    ModPerl::MethodLookup::lookup_method('print');

# return only module names containing method 'print' which
# expects the first argument to be of type 'Apache::Filter'
# (here $filter is an Apache::Filter object)
my($hint, @modules) =
    ModPerl::MethodLookup::lookup_method('print', $filter);
# or
my($hint, @modules) =
    ModPerl::MethodLookup::lookup_method('print', 'Apache::Filter');

# what XS methods defined by module 'Apache::Filter'
my($hint, @methods) =
    ModPerl::MethodLookup::lookup_module('Apache::Filter');

# what XS methods can be invoked on the object $r (or a ref)
my($hint, @methods) =
    ModPerl::MethodLookup::lookup_object($r);
# or
my($hint, @methods) =
    ModPerl::MethodLookup::lookup_object('Apache::RequestRec');

# preload all mp2 modules in startup.pl
ModPerl::MethodLookup::preload_all_modules();

# command line shortcuts
% perl -MApache2 -MModPerl::MethodLookup -e print_module \
    Apache::RequestRec Apache::Filter
% perl -MApache2 -MModPerl::MethodLookup -e print_object Apache
% perl -MApache2 -MModPerl::MethodLookup -e print_method \
    get_server_built request
% perl -MApache2 -MModPerl::MethodLookup -e print_method read
% perl -MApache2 -MModPerl::MethodLookup -e print_method read APR::Bucket

```

42.2 Description

mod_perl 2.0 provides many methods, which reside in various modules. One has to load each of the modules before using the desired methods. `ModPerl::MethodLookup` provides the Perl API for finding module names which contain methods in question and other helper functions, like figuring out what methods defined by some module, or what methods can be called on a given object.

42.3 API

42.3.1 *lookup_method()*

```
my($hint, @modules) =
    ModPerl::MethodLookup::lookup_method($method_name);
```

The `lookup_method()` function accepts the method name as the first argument.

The first returned value is a string returning a human readable lookup result. Normally suggesting which modules should be loaded, ready for copy-n-paste or explaining the failure if the lookup didn't succeed.

The second returned value is an array of modules which have matched the query, i.e. the names of the modules which contain the requested method.

```
my($hint, @modules) =
    ModPerl::MethodLookup::lookup_method($method_name, $object);
```

or:

```
my($hint, @modules) =
    ModPerl::MethodLookup::lookup_method($method_name, ref($object));
```

The `lookup_method()` function accepts a second optional argument, which is a blessed object or the class it's blessed into. If there is more than one matches this extra information is used to return only modules of those methods which operate on the objects of the same kind. This usage is useful when the `AUTOLOAD` is used to find yet-unloaded modules which include called methods.

Examples:

Return all module names containing XS method *print*:

```
my($hint, @modules) =
    ModPerl::MethodLookup::lookup_method('print');
```

Return only module names containing method *print* which expects the first argument to be of type `Apache::Filter`:

```
my $filter = bless {}, 'Apache::Filter';
my($hint, @modules) =
    ModPerl::MethodLookup::lookup_method('print', $filter);
```

or:

```
my($hint, @modules) =
    ModPerl::MethodLookup::lookup_method('print', 'Apache::Filter');
```

42.3.2 *lookup_module()*

```
my($hint, @methods) =
    ModPerl::MethodLookup::lookup_module($module_name);
```

The `lookup_module()` function accepts the module name as an argument.

The first returned value is a string returning a human readable lookup result. Normally suggesting, which methods the given module implements, or explaining the failure if the lookup didn't succeed.

The second returned value is an array of methods which have matched the query, i.e. the names of the methods defined in the requested module.

Example:

What XS methods defined by module `Apache::Filter`:

```
my($hint, @methods) =
    ModPerl::MethodLookup::lookup_module('Apache::Filter');
```

42.3.3 *lookup_object()*

```
my($hint, @methods) =
    ModPerl::MethodLookup::lookup_object($object);
```

or:

```
my($hint, @methods) =
    ModPerl::MethodLookup::lookup_object(
        $the_class_object_is_blessed_into);
```

The `lookup_object()` function accepts the object or the class name the object is blessed into as an argument.

The first returned value is a string returning a human readable lookup result. Normally suggesting, which methods the given object can invoke (including module names that need to be loaded to use those methods), or explaining the failure if the lookup didn't succeed.

The second returned value is an array of methods which have matched the query, i.e. the names of the methods that can be invoked on the given object (or its class name).

META: As of this writing this method may miss some of the functions/methods that can be invoked on the given object. Currently we can't programmatically deduct the objects they are invoked on, because these methods are written in pure XS and manipulate the arguments stack themselves. Currently these are mainly XS functions, not methods, which of course aren't invoked on objects. There are also logging function wrappers (`Apache::Log`).

Examples:

What XS methods can be invoked on the object `$r`:

```
my($hint, @methods) =
    ModPerl::MethodLookup::lookup_object($r);
```

or `$r`'s class -- `Apache::RequestRec`:

```
my($hint, @methods) =
    ModPerl::MethodLookup::lookup_object('Apache::RequestRec');
```

42.3.4 *print_method()*

`print_method()` is a convenience wrapper for `lookup_method()`, mainly designed to be used from the command line. For example to print all the modules which define method *read* execute:

```
% perl -MApache2 -MModPerl::MethodLookup -e print_method read
```

Since this will return more than one module, we can narrow the query to only those methods which expect the first argument to be blessed into class `APR::Bucket`:

```
% perl -MApache2 -MModPerl::MethodLookup -e print_method read APR::Bucket
```

You can pass more than one method and it'll perform a lookup on each of the methods. For example to lookup methods `get_server_built` and `request` you can do:

```
% perl -MApache2 -MModPerl::MethodLookup -e print_method \
    get_server_built request
```

The function `print_method()` is exported by default.

42.3.5 *print_module()*

`print_module()` is a convenience wrapper for `lookup_module()`, mainly designed to be used from the command line. For example to print all the methods defined in the module `Apache::RequestRec`, followed by methods defined in the module `Apache::Filter` you can run:

```
% perl -MApache2 -MModPerl::MethodLookup -e print_module \
    Apache::RequestRec Apache::Filter
```

The function `print_module()` is exported by default.

42.3.6 *print_object()*

`print_object()` is a convenience wrapper for `lookup_object()`, mainly designed to be used from the command line. For example to print all the methods that can be invoked on object blessed into a class `Apache::RequestRec` run:

```
% perl -MApache2 -MModPerl::MethodLookup -e print_object \
    Apache::RequestRec
```

Similar to `print_object()`, more than one class can be passed to this function.

The function `print_object()` is exported by default.

42.3.7 *preload_all_modules()*

The function `preload_all_modules()` preloads all `mod_perl` 2.0 modules, which implement their API in XS. This is similar to the `mod_perl` 1.0 behavior which has most of its methods loaded at the startup.

CPAN modules developers should make sure their distribution loads each of the used `mod_perl` 2.0 modules explicitly, and not use this function, as it takes the fine control away from the users. One should avoid doing this the production server (unless all modules are used indeed) in order to save memory.

42.4 Applications

42.4.1 *AUTOLOAD*

When Perl fails to locate a method it checks whether the package the object belongs to has an `AUTOLOAD` function defined and if so, calls it with the same arguments as the missing method while setting a global variable `$AUTOLOAD` (in that package) to the name of the originally called method. We can use this facility to lookup the modules to be loaded when such a failure occurs. Though since we have many packages to take care of we will use a special `UNIVERSAL::AUTOLOAD` function which Perl calls if can't find the `AUTOLOAD` function in the given package.

In that function you can query `ModPerl::MethodLookup`, `require()` the module that includes the called method and call that method again using the `goto()` trick:

```
use ModPerl::MethodLookup;
sub UNIVERSAL::AUTOLOAD {
    my($hint, @modules) =
        ModPerl::MethodLookup::lookup_method($UNIVERSAL::AUTOLOAD, @_);
    if (@modules) {
        eval "require $_" for @modules;
        goto &$UNIVERSAL::AUTOLOAD;
    }
    else {
        die $hint;
    }
}
```

However we don't endorse this approach. It's a better approach to always abort the execution which printing the `$hint` and use fix the code to load the missing module. Moreover installing `UNIVERSAL::AUTOLOAD` may cause a lot of problems, since once it's installed Perl will call it every time some method is missing (e.g. undefined `DESTROY` methods). The following approach seems to somewhat work for me. It installs `UNIVERSAL::AUTOLOAD` only when the the child process starts.

```

httpd.conf:
-----
PerlChildInitHandler ModPerl::MethodLookupAuto

startup.pl:
-----
{
    package ModPerl::MethodLookupAuto;
    use ModPerl::MethodLookup;

    use Carp;
    sub handler {

        # exclude DESTROY resolving
        my $skip = '^(?!DESTROY$';
        *UNIVERSAL::AUTOLOAD = sub {
            my $method = $AUTOLOAD;
            return if $method =~ /DESTROY/;
            my ($hint, @modules) =
                ModPerl::MethodLookup::lookup_method($method, @_);
            $hint ||= "Can't find method $AUTOLOAD";
            croak $hint;
        };
        return 0;
    }
}

```

This example doesn't load the modules for you. It'll print to `STDERR` what module should be loaded, when a method from the not-yet-loaded module is called.

A similar technique is used by `Apache::porting`.

42.4.2 Command Line Lookups

When a method is used and `mod_perl` has reported a failure to find it, it's often useful to use the command line query to figure out which module needs to be loaded. For example if when executing:

```
$r->construct_url();
```

`mod_perl` complains:

```
Can't locate object method "construct_url" via package
"Apache::RequestRec" at ...
```

you can ask `ModPerl::MethodLookup` for help:

```
% perl -MApache2 -MModPerl::MethodLookup -e print_method construct_url
To use method 'construct_url' add:
    use Apache::URI ();
```

and after copy-n-pasting the use statement in our code, the problem goes away.

One can create a handy alias for this technique. For example, C-style shell users can do:

```
% alias lookup "perl -MApache2 -MModPerl::MethodLookup -e print_method"
```

For Bash-style shell users:

```
% alias lookup="perl -MApache2 -MModPerl::MethodLookup -e print_method"
```

Now the lookup is even easier:

```
% lookup construct_url
to use method 'construct_url' add:
    use Apache::URI;
```

Similar aliases can be provided for `print_object()` and `print_module()`.

42.5 Todo

These methods aren't yet picked by this module (the extract from the map file):

<code>modperl_filter_attributes</code>		<code>MODIFY_CODE_ATTRIBUTES</code>
<code>modperl_spawn_proc_prog</code>		<code>spawn_proc_prog</code>
<code>apr_ipsubnet_create</code>		<code>new</code>

Please report if you find any other missing methods. But remember that as of this moment the module reports only XS function. In the future we may add support for pure perl functions/methods as well.

42.6 See Also

- the `mod_perl` 1.0 backward compatibility document
- porting Perl modules
- porting XS modules
- `Apache::porting`

42.7 Author

Stas Bekman

43 ModPerl::MM -- A "subclass" of ExtUtils::MakeMaker for mod_perl 2.0

43.1 Synopsis

```
use ModPerl::MM;

# ModPerl::MM takes care of doing all the dirty job of overriding
ModPerl::MM::WriteMakefile(...);

# if there is a need to extend the default methods
sub MY::constants {
    my $self = shift;
    $self->ModPerl::MM::MY::constants;
    # do something else;
}

# or prevent overriding completely
sub MY::constants { shift->MM::constants(@_); };

# override the default value of WriteMakefile's attribute
my $extra_inc = "/foo/include";
ModPerl::MM::WriteMakefile(
    ...
    INC => $extra_inc,
    ...
);

# extend the default value of WriteMakefile's attribute
my $extra_inc = "/foo/include";
ModPerl::MM::WriteMakefile(
    ...
    INC => join " ", $extra_inc, ModPerl::MM::get_def_opt('INC'),
    ...
);
```

43.2 Description

ModPerl::MM is a "subclass" of ExtUtils::MakeMaker for mod_perl 2.0, to a degree of sub-classability of ExtUtils::MakeMaker.

When ModPerl::MM::WriteMakefile() is used instead of ExtUtils::MakeMaker::WriteMakefile(), ModPerl::MM overrides several ExtUtils::MakeMaker methods behind the scenes and supplies default WriteMakefile() arguments adjusted for mod_perl 2.0 build. It's written in such a way so that normally 3rd party module developers for mod_perl 2.0, don't need to mess with *Makefile.PL* at all.

43.3 MY::: Default Methods

ModPerl::MM overrides method *foo* as long as *Makefile.PL* hasn't already specified a method *MY::foo*. If the latter happens, ModPerl::MM will DWIM and do nothing.

In case the functionality of `ModPerl::MM` methods needs to be extended, rather than completely overridden, the `ModPerl::MM` methods can be called internally. For example if you need to modify constants in addition to the modifications applied by `ModPerl::MM::MY::constants`, call the `ModPerl::MM::MY::constants` method (notice that it resides in the package `ModPerl::MM::MY` and not `ModPerl::MM`), then do your extra manipulations on constants:

```
# if there is a need to extend the methods
sub MY::constants {
    my $self = shift;
    $self->ModPerl::MM::MY::constants;
    # do something else;
}
```

In certain cases a developers may want to prevent from `ModPerl::MM` to override certain methods. In that case an explicit override in *Makefile.PL* will do the job. For example if you don't want the `constants()` method to be overridden by `ModPerl::MM`, add to your *Makefile.PL*:

```
sub MY::constants { shift->MM::constants(@_); }";
```

`ModPerl::MM` overrides the following methods:

43.3.1 *ModPerl::MM::MY::constants*

This method makes sure that everything gets installed relative to the `Apache2/` subdir if `MP_INST_APACHE2=1` was used to build `mod_perl 2.0`.

43.3.2 *ModPerl::MM::MY::post_initialize*

This method makes sure that everything gets installed relative to the `Apache2/` subdir if `MP_INST_APACHE2=1` was used to build `mod_perl 2.0`.

43.4 `WriteMakefile()` Default Arguments

`ModPerl::MM::WriteMakefile` supplies default arguments such as `INC` and `TYPEMAPS` unless they weren't passed to `ModPerl::MM::WriteMakefile` from *Makefile.PL*.

If the default values aren't satisfying these should be overridden in *Makefile.PL*. For example to supply an empty `INC`, explicitly set the argument in *Makefile.PL*.

```
ModPerl::MM::WriteMakefile(
    ...
    INC => '',
    ...
);
```

If instead of fully overriding the default arguments, you want to extend or modify them, they can be retrieved using the `ModPerl::MM::get_def_opt()` function. The following example appends an extra value to the default `INC` attribute:

43.5 Public API

```
my $extra_inc = "/foo/include";
ModPerl::MM::WriteMakefile(
    ...
    INC => join " ", $extra_inc, ModPerl::MM::get_def_opt('INC'),
    ...
);
```

ModPerl::MM supplies default values for the following ModPerl::MM::WriteMakefile attributes:

43.4.1 CCFLAGS

43.4.2 LIBS

43.4.3 INC

43.4.4 OPTIMIZE

43.4.5 LDDLFLAGS

43.4.6 TYPEMAPS

43.4.7 dynamic_lib

43.4.7.1 OTHERLDFLAGS

```
dynamic_lib => { OTHERLDFLAGS => ... }
```

43.4.8 macro

43.4.8.1 MOD_INSTALL

```
macro => { MOD_INSTALL => ... }
```

arranges for modules to be installed under the subdir *Apache2/* if *mod_perl* was built with *MP_INST_APACHE2=1*.

43.5 Public API

The following functions are a part of the public API. They are described elsewhere in this document.

43.5.1 WriteMakefile()

```
ModPerl::MM::WriteMakefile(...);
```

43.5.2 get_def_opt()

```
my $def_val = ModPerl::MM::get_def_opt($key);
```

44 ModPerl::PerlRun - Run unaltered CGI scripts under mod_perl

44.1 SYNOPSIS

44.2 DESCRIPTION

44.3 AUTHORS

Doug MacEachern

Stas Bekman

44.4 SEE ALSO

ModPerl::RegistryCooker(3), Apache(3), mod_perl(3)

45 ModPerl::Registry - Run unaltered CGI scripts persistently under mod_perl

45.1 Synopsis

```
# httpd.conf
PerlModule ModPerl::Registry
Alias /perl/ /home/httpd/perl/
<Location /perl>
    SetHandler perl-script
    PerlResponseHandler ModPerl::Registry
    #PerlOptions +ParseHeaders
    #PerlOptions -GlobalRequest
    Options +ExecCGI
</Location>
```

45.2 Description

URIs in the form of `http://example.com/perl/test.pl` will be compiled as the body of a Perl subroutine and executed. Each child process will compile the subroutine once and store it in memory. It will recompile it whenever the file (e.g. *test.pl* in our example) 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 into a subroutine.

For example:

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

XXX: STOPPED here. Below is the old Apache::Registry document which I haven't worked through yet.

META: document that for now we don't `chdir()` into the script's dir, because it affects the whole process under threads.

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 `ModPerl::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.

45.3 Security

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

45.4 Environment

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

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

45.5 Commandline Switches In 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.

ModPerl::Registry currently only honors the **-w** switch and will enable the warnings pragma in such case.

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:

```
PerlSwitches -T
```

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

45.6 Debugging

You may set the debug level with the \$ModPerl::Registry::Debug bitmask

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

45.7 Caveats

ModPerl::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. (META: works in 2.0).

45.8 Authors

Andreas J. Koenig, Doug MacEachern and Stas Bekman.

45.9 See Also

ModPerl::RegistryCooker, ModPerl::RegistryBB, ModPerl::PerlRun, Apache(3), mod_perl(3)

46 ModPerl::RegistryBB - Run unaltered CGI scripts persistently under mod_perl

46.1 Synopsis

```
# httpd.conf
PerlModule ModPerl::RegistryBB
Alias /perl/ /home/httpd/perl/
<Location /perl>
    SetHandler perl-script
    PerlResponseHandler ModPerl::RegistryBB
    #PerlOptions +ParseHeaders
    #PerlOptions -GlobalRequest
    Options +ExecCGI
</Location>
```

46.2 Description

ModPerl::RegistryBB is similar to ModPerl::Registry, but does the bare minimum (mnemonic: BB = Bare Bones) to compile a script file once and run it many times, in order to get the maximum performance. Whereas ModPerl::Registry does various checks, which add a slight overhead to response times.

46.3 Authors

Doug MacEachern

Stas Bekman

46.4 See Also

ModPerl::RegistryCooker, ModPerl::Registry, Apache(3), mod_perl(3)

47 ModPerl::RegistryCooker - Cook mod_perl 2.0 Registry Modules

47.1 Synopsis

```
# shouldn't be used as-is but sub-classed first
# see ModPerl::Registry for an example
```

47.2 Description

ModPerl::RegistryCooker is used to create flexible and overridable registry modules which emulate mod_cgi for Perl scripts. The concepts are discussed in the manpage of the following modules: ModPerl::Registry, ModPerl::Registry and ModPerl::RegistryBB.

ModPerl::RegistryCooker has two purposes:

- Provide ingredients that can be used by registry sub-classes
- Provide a default behavior, which can be overridden in sub-classed

META: in the future this functionality may move into a separate class.

Here are the current overridable methods:

META: these are all documented in RegistryCooker.pm, though not using pod. please help to port these to pod and move the descriptions here.

- **new()**

create the class's object, bless it and return it

```
my $obj = $class->new($r);
```

\$class -- the registry class, usually `__PACKAGE__` can be used.

\$r -- *Apache::Request* object.

default: new()

- **init()**

initializes the data object's fields: REQ, FILENAME, URI. Called from the new().

default: init()

- **default_handler()**

default: default_handler()

- **run()**

default: run()

- **can_compile()**

default: can_compile()

- **make_namespace()**

default: make_namespace()

- **namespace_root()**

default: namespace_root()

- **namespace_from()**

If `namespace_from_uri` is used and the script is called from the virtual host, by default the virtual host name is prepended to the uri when package name for the compiled script is created. Sometimes this behavior is undesirable, e.g., when the same (physical) script is accessed using the same `path_info` but different virtual hosts. In that case you can make the script compiled only once for all vhosts, by specifying:

```
$ModPerl::RegistryCooker::NameWithVirtualHost = 0;
```

The drawback is that it affects the global environment and all other scripts will be compiled ignoring virtual hosts.

default: namespace_from()

- **is_cached()**

default: is_cached()

- **should_compile()**

default: should_compile()

- **flush_namespace()**

default: flush_namespace()

- **cache_table()**

default: cache_table()

- **cache_it()**

default: cache_it()

- **read_script()**
default: read_script()
- **rewrite_shebang()**
default: rewrite_shebang()
- **get_script_name()**
default: get_script_name()
- **chdir_file()**
default: chdir_file()
- **get_mark_line()**
default: get_mark_line()
- **compile()**
default: compile()
- **error_check()**
default: error_check()
- **strip_end_data_segment()**
default: strip_end_data_segment()
- **convert_script_to_compiled_handler()**
default: convert_script_to_compiled_handler()

47.2.1 Special Predefined Functions

The following functions are implemented as constants.

- **NOP()**
Use when the function shouldn't do anything.
- **TRUE()**
Use when a function should always return a true value.

- **FALSE()**

Use when a function should always return a false value.

47.3 Sub-classing Techniques

To override the default `ModPerl::RegistryCooker` methods, first, sub-class `ModPerl::RegistryCooker` or one of its existing sub-classes, using `use base`. Second, override the methods.

Those methods that weren't overridden will be resolved at run time when used for the first time and cached for the future requests. One way to shortcut this first run resolution is to use the symbol aliasing feature. For example to alias `ModPerl::MyRegistry::flush_namespace` as `ModPerl::RegistryCooker::flush_namespace`, you can do:

```
package ModPerl::MyRegistry;
use base qw(ModPerl::RegistryCooker);
*ModPerl::MyRegistry::flush_namespace =
    \&ModPerl::RegistryCooker::flush_namespace;
1;
```

In fact, it's a good idea to explicitly alias all the methods so you know exactly what functions are used, rather than relying on the defaults. For that purpose `ModPerl::RegistryCooker` class method `install_aliases()` can be used. Simply prepare a hash with method names in the current package as keys and corresponding fully qualified methods to be aliased for as values and pass it to `install_aliases()`. Continuing our example we could do:

```
package ModPerl::MyRegistry;
use base qw(ModPerl::RegistryCooker);
my %aliases = (
    flush_namespace => 'ModPerl::RegistryCooker::flush_namespace',
);
__PACKAGE__->install_aliases(\%aliases);
1;
```

The values use fully qualified packages so you can mix methods from different classes.

47.4 Examples

The best examples are existing core registry modules: `ModPerl::Registry`, `ModPerl::Registry` and `ModPerl::RegistryBB`. Look at the source code and their manpages to see how they subclass `ModPerl::RegistryCooker`.

For example by default `ModPerl::Registry` uses the script's path when creating a package's namespace. If for example you want to use a uri instead you can override it with:

```
*ModPerl::MyRegistry::namespace_from =
    \&ModPerl::RegistryCooker::namespace_from_uri;
1;
```

Since the `namespace_from_uri` component already exists in `ModPerl::RegistryCooker`. If you want to write your own method, e.g., that creates a namespace based on the inode, you can do:

```
sub namespace_from_inode {  
    my $self = shift;  
    return (stat $self->[FILENAME])[1];  
}
```

META: when `$r->finfo` will be ported it'll be more effecient. `(stat $r->finfo)[1]`

47.5 Authors

Doug MacEachern

Stas Bekman

47.6 See Also

`ModPerl::Registry`, `ModPerl::RegistryBB`, `ModPerl::PerlRun`, `Apache(3)`, `mod_perl(3)`

48 ModPerl::RegistryLoader - Compile ModPerl::RegistryCooker scripts at server startup

48.1 Synopsis

```
# in startup.pl
use ModPerl::RegistryLoader ();
use APR::Pool ();

# explicit uri => filename mapping
my $rlbb = ModPerl::RegistryLoader->new(
    package => 'ModPerl::RegistryBB',
    debug   => 1, # default 0
);

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

###
# uri => filename mapping using a helper function
sub trans {
    my $uri = shift;
    $uri =~ s|^/registry/|cgi-bin/|;
    return Apache::Server::server_root_relative(APR::Pool->new, $uri);
}
my $rl = ModPerl::RegistryLoader->new(
    package => 'ModPerl::Registry',
    trans   => \&trans,
);
$rl->handler($uri);

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

48.2 Description

This module allows compilation of scripts, running under packages derived from `ModPerl::RegistryCooker`, at server startup. The script's handler routine is compiled by the parent server, of which children get a copy and thus saves some memory by initially sharing the compiled copy with the parent and saving the overhead of script's compilation on the first request in every httpd instance.

This module is of course useless for those running the `ModPerl::PerlRun` handler, because the scripts get recompiled on each request under this handler.

48.3 Methods

- **new()**

When creating a new `ModPerl::RegistryLoader` object, one has to specify which of the `ModPerl::RegistryCooker` derived modules to use. For example if a script is going to run under `ModPerl::RegistryBB` the object is initialized as:

```
my $rlbb = ModPerl::RegistryLoader->new(
    package => 'ModPerl::RegistryBB',
);
```

If the package is not specified `ModPerl::Registry` is assumed:

```
my $rlbb = ModPerl::RegistryLoader->new();
```

To turn the debugging on, set the *debug* attribute to a true value:

```
my $rlbb = ModPerl::RegistryLoader->new(
    package => 'ModPerl::RegistryBB',
    debug    => 1,
);
```

Instead of specifying explicitly a filename for each uri passed to `handler()`, a special attribute *trans* can be set to a subroutine to perform automatic remapping.

```
my $rlbb = ModPerl::RegistryLoader->new(
    package => 'ModPerl::RegistryBB',
    trans    => \&trans,
);
```

See the `handler()` item for an example of using the *trans* attribute.

● **handler()**

```
$rl->handler($uri, [$filename, [$virtual_hostname]]);
```

The `handler()` method takes argument of uri and optionally of filename and of `virtual_hostname`.

URI to filename translation normally doesn't happen until HTTP request time, so we're forced to roll our own translation. If the filename is supplied it's used in translation.

If the filename is omitted and a `trans` subroutine was not set in `new()`, the loader will try using the uri relative to the `ServerRoot` configuration directive. For example:

```
httpd.conf:
-----
ServerRoot /usr/local/apache
Alias /registry/ /usr/local/apache/cgi-bin/

startup.pl:
-----
use ModPerl::RegistryLoader ();
my $rl = ModPerl::RegistryLoader->new(
    package => 'ModPerl::Registry',
);
# preload /usr/local/apache/cgi-bin/test.pl
$rl->handler(/registry/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 `cgi-bin` directory relative to `ServerRoot`.

```
httpd.conf:
-----
ServerRoot /usr/local/apache
Alias /registry/ /usr/local/apache/cgi-bin/

startup.pl:
-----
{
    # test the scripts pre-loading by using trans sub
    use ModPerl::RegistryLoader ();
    use APR::Pool ();
    use DirHandle ();
    use strict;

    my $pool = APR::Pool->new;

    my $dir = Apache::Server::server_root_relative($pool, "cgi-bin");

    sub trans {
        my $uri = shift;
        $uri =~ s|^/registry/|cgi-bin/|;
        return Apache::Server::server_root_relative($pool, $uri);
    }

    my $rl = ModPerl::RegistryLoader->new(
        package => "ModPerl::Registry",
        trans    => \&trans,
    );
    my $dh = DirHandle->new($dir) or die $!;

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

If `$virtual_hostname` argument is passed it'll be used in the creation of the package name the script will be compiled into for those registry handlers that use `namespace_from_uri()` method. See also the notes on `$ModPerl::RegistryCooker::NameWithVirtualHost` in the `ModPerl::RegistryCooker` documentation.

Also explained in the `ModPerl::RegistryLoader` documentation, this only has an effect at run time if `$ModPerl::RegistryCooker::NameWithVirtualHost` is set to true, otherwise the `$virtual_hostname` argument is ignored.

48.4 Implementation Notes

`ModPerl::RegistryLoader` performs a very simple job, at run time it loads and sub-classes the module passed via the *package* attribute and overrides some of its functions, to emulate the run-time environment. This allows to preload the same script into different registry environments.

48.5 Authors

The original `Apache::RegistryLoader` implemented by Doug MacEachern.

Stas Bekman did the porting to the new registry framework based on `ModPerl::RegistryLoader`.

48.6 SEE ALSO

`ModPerl::RegistryCooker`, `ModPerl::Registry`, `ModPerl::RegistryBB`,
`ModPerl::PerlRun`, `Apache(3)`, `mod_perl(3)`

49 ModPerl::Util - Helper mod_perl 2.0 Functions

49.1 Synopsis

```
use ModPerl::Util;

$callback = Apache::current_callback;

ModPerl::Util::exit();

ModPerl::Util::untaint($) # secret API?
```

49.2 Description

ModPerl::Util provides mod_perl 2.0 util functions.

49.3 API

ModPerl::Util provides the following functions and/or methods:

49.3.1 *untaint*

Untaint the variable, by turning its tainted bit off (used internally).

```
untaint($var);
```

- **arg1: \$var** (scalar)
- **ret: no return value**

Do not use this function unless you know what you are doing. To learn how to properly untaint variables refer to the *perlsec* manpage.

49.3.2 *current_callback*

Returns the currently running callback, like 'PerlResponseHandler'.

```
$callback = Apache::current_callback();
```

- **ret: \$callback** (string)

49.3.3 *exit*

Used internally to replace `CORE::exit` and terminate the request, not the current process.

```
ModPerl::Util::exit($status);
```

- **opt arg1: \$status** (integer)

The exit status, similar to `CORE::exit`. If not passed, the default value of 0 is used.

- **ret: no return value**

49.4 See Also

mod_perl 2.0 documentation.

49.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

49.6 Authors

The mod_perl development team and numerous contributors.

50 Apache::porting -- a helper module for mod_perl 1.0 to mod_perl 2.0 porting

50 Apache::porting -- a helper module for mod_perl 1.0 to mod_perl 2.0 porting

50.1 Synopsis

```
# either add at the very beginning of startup.pl
use Apache2;
use Apache::porting;

# or httpd.conf
PerlModule Apache2
PerlModule Apache::porting

# now issue requests and look at the error_log file for hints
```

50.2 Description

Apache::porting helps to port mod_perl 1.0 code to run under mod_perl 2.0. It doesn't provide any back-compatibility functionality, however it knows trap calls to methods that are no longer in the mod_perl 2.0 API and tell what should be used instead if at all. If you attempts to use mod_perl 2.0 methods without first loading the modules that contain them, it will tell you which modules you need to load. Finally if your code tries to load modules that no longer exist in mod_perl 2.0 it'll also tell you what are the modules that should be used instead.

Apache::porting communicates with users via the *error_log* file. Everytime it traps a problem, it logs the solution (if it finds one) to the error log file. If you use this module coupled with Apache::Reload you will be able to port your applications quickly without needing to restart the server on every modification.

It starts to work only when child process start and doesn't work for the code that gets loaded at the server startup. This limitation is explained in the Culprits section.

It relies heavily on `ModPerl::MethodLookup`, which can also be used manually to lookup things.

50.3 Culprits

Apache::porting uses the `UNIVERSAL::AUTOLOAD` function to provide its functionality. However it seems to be impossible to create `UNIVERSAL::AUTOLOAD` at the server startup, Apache segfaults on restart. Therefore it performs the setting of `UNIVERSAL::AUTOLOAD` only during the *child_init* phase, when child processes start. As a result it can't help you with things that get preloaded at the server startup.

If you know how to resolve this problem, please let us know. To reproduce the problem try to use an earlier phase, e.g. `PerlPostConfigHandler`:

```
Apache->server->push_handlers(PerlPostConfigHandler => \&porting_autoload);
```

50.4 See Also

`mod_perl` 2.0 documentation.

50.5 Copyright

`mod_perl` 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

50.6 Authors

The `mod_perl` development team and numerous contributors.

51 Apache::Reload - Reload Perl Modules when Changed on Disk

51.1 Synopsis

```
# Monitor and reload all modules in %INC:
# httpd.conf:
PerlModule Apache::Reload
PerlInitHandler Apache::Reload

# when working with protocols and connection filters
# PerlPreConnectionHandler Apache::Reload

# Reload groups of modules:
# httpd.conf:
PerlModule Apache::Reload
PerlInitHandler Apache::Reload
PerlSetVar ReloadAll Off
PerlSetVar ReloadModules "ModPerl:* Apache:*"
#PerlSetVar ReloadDebug On
#PerlSetVar ReloadConstantRedefineWarnings Off

# Reload a single module from within itself:
package My::Apache::Module;
use Apache::Reload;
sub handler { ... }
1;
```

51.2 Description

`Apache::Reload` reloads modules that change on the 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 can be configured to iterate over the modules in `%INC` and reload those that have changed on disk or only specific modules that have registered themselves with `Apache::Reload`. It can also do the check for modified modules, when a special touch-file has been modified.

Note that `Apache::Reload` 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 `Apache::Reload` to work with modules that live in custom `@INC` paths, you should modify `@INC` when the server is started. Besides, `'use lib'` in the startup script, you can also set the `PERL5LIB` variable in the `httpd`'s environment to include any non-standard `'lib'` directories that you choose. For example, to accomplish that you can include a line:

```
PERL5LIB=/home/httpd/perl/extra; export PERL5LIB
```

in the script that starts Apache. Alternatively, you can set this environment variable in *httpd.conf*:

```
PerlSetEnv PERL5LIB /home/httpd/perl/extra
```


51.2.1 Monitor All Modules in %INC

To monitor and reload all modules in %INC at the beginning of request's processing, simply add the following configuration to your *httpd.conf*:

```
PerlModule Apache::Reload
PerlInitHandler Apache::Reload
```

When working with connection filters and protocol modules Apache::Reload should be invoked in the *pre_connection* stage:

```
PerlPreConnectionHandler Apache::Reload
```

See also the discussion on *PerlPreConnectionHandler*.

51.2.2 Register Modules Implicitly

To only reload modules that have registered with Apache::Reload, add the following to the *httpd.conf*:

```
PerlModule Apache::Reload
PerlInitHandler Apache::Reload
PerlSetVar ReloadAll Off
# ReloadAll defaults to On
```

Then any modules with the line:

```
use Apache::Reload;
```

Will be reloaded when they change.

51.2.3 Register Modules Explicitly

You can also register modules explicitly in your *httpd.conf* file that you want to be reloaded on change:

```
PerlModule Apache::Reload
PerlInitHandler Apache::Reload
PerlSetVar ReloadAll Off
PerlSetVar ReloadModules "My::Foo My::Bar Foo::Bar::Test"
```

Note that these are split on whitespace, but the module list **must** be in quotes, otherwise Apache tries to parse the parameter list.

The *** wild character can be used to register groups of files under the same namespace. For example the setting:

```
PerlSetVar ReloadModules "ModPerl::* Apache::*"
```

will monitor all modules under the namespaces `ModPerl::` and `Apache::`.

51.2.4 Monitor Only Certain Sub Directories

To reload modules only in certain directories (and their subdirectories) add the following to the *httpd.conf*:

```
PerlModule Apache::Reload
PerlInitHandler Apache::Reload
PerlSetVar ReloadDirectories "/tmp/project1 /tmp/project2"
```

You can further narrow the list of modules to be reloaded from the chosen directories with `ReloadModules` as in:

```
PerlModule Apache::Reload
PerlInitHandler Apache::Reload
PerlSetVar ReloadDirectories "/tmp/project1 /tmp/project2"
PerlSetVar ReloadAll Off
PerlSetVar ReloadModules "MyApache::"
```

In this configuration example only modules from the namespace `MyApache::` found in the directories */tmp/project1/* and */tmp/project2/* (and their subdirectories) will be reloaded.

51.2.5 Special "Touch" File

You can also declare a file, which when gets `touch(1)`ed, causes the reloads to be performed. For example if you set:

```
PerlSetVar ReloadTouchFile /tmp/reload_modules
```

and don't `touch(1)` the file */tmp/reload_modules*, the reloads won't happen until you go to the command line and type:

```
% touch /tmp/reload_modules
```

When you do that, the modules that have been changed, will be magically reloaded on the next request. This option works with any mode described before.

51.3 Performance Issues

This modules is perfectly suited for a development environment. Though it's possible that you would like to use it in a production environment, since with `Apache::Reload` you don't have to restart the server in order to reload changed modules during software updates. Though this convenience comes at a price:

- If the "touch" file feature is used, `Apache::Reload` has to `stat(2)` the touch file on each request, which adds a slight but most likely insignificant overhead to response times. Otherwise `Apache::Reload` will `stat(2)` each registered module or even worse--all modules in `%INC`, which will significantly slow everything down.

- Once the child process reloads the modules, the memory used by these modules is not shared with the parent process anymore. Therefore the memory consumption may grow significantly.

Therefore doing a full server stop and restart is probably a better solution.

51.4 Debug

If you aren't sure whether the modules that are supposed to be reloaded, are actually getting reloaded, turn the debug mode on:

```
PerlSetVar ReloadDebug On
```

51.5 Silencing 'Constant subroutine ... redefined at' Warnings

If a module defines constants, e.g.:

```
use constant PI => 3.14;
```

and gets re-loaded, Perl issues a mandatory warnings which can't be silenced by conventional means (since Perl 5.8.0). This is because constants are inlined at compile time, so if there are other modules that are using constants from this module, but weren't reloaded they will see different values. Hence the warning is mandatory. However chances are that most of the time you won't modify the constant subroutine and you don't want *error_log* to be cluttered with (hopefully) irrelevant warnings. In such cases, if you haven't modified the constant subroutine, or you know what you are doing, you can tell Apache::Reload to shut those for you (it overrides `$SIG{__WARN__}` to accomplish that):

```
PerlSetVar ReloadConstantRedefineWarnings Off
```

For the reasons explained above this option is turned on by default.

since: mod_perl 1.99_10

51.6 Caveats

51.6.1 Problems With Reloading Modules Which Do Not Declare Their Package Name

If you modify modules, which don't declare their package, and rely on Apache::Reload to reload them, you may encounter problems: i.e., it'll appear as if the module wasn't reloaded when in fact it was. This happens because when Apache::Reload `require()`s such a module all the global symbols end up in the Apache::Reload namespace! So the module does get reloaded and you see the compile time errors if there are any, but the symbols don't get imported to the right namespace. Therefore the old version of the code is running.

51.6.2 Problems with Scripts Running with Registry Handlers that Cache the Code

The following problem is relevant only to registry handlers that cache the compiled script. For example it concerns `ModPerl::Registry` but not `ModPerl::PerlRun`.

51.6.2.1 The Problem

Let's say that there is a module `My::Utils`:

```
#file:My/Utils.pm
#-----
package My::Utils;
BEGIN { warn __PACKAGE__ , " was reloaded\n" }
use base qw(Exporter);
@EXPORT = qw(colour);
sub colour { "white" }
1;
```

And a registry script *test.pl*:

```
#file:test.pl
#-----
use My::Utils;
print "Content-type: text/plain\n\n";
print "the color is " . colour();
```

Assuming that the server is running in a single mode, we request the script for the first time and we get the response:

```
the color is white
```

Now we change *My/Utils.pm*:

```
- sub colour { "white" }
+ sub colour { "red" }
```

And issue the request again. `Apache::Reload` does its job and we can see that `My::Utils` was reloaded (look in the *error_log* file). However the script still returns:

```
the color is white
```

51.6.2.2 The Explanation

Even though *My/Utils.pm* was reloaded, `ModPerl::Registry`'s cached code won't run `'use My::Utils;'` again (since it happens only once, i.e. during the compile time). Therefore the script doesn't know that the subroutine reference has been changed.

This is easy to verify. Let's change the script to be:

```
#file:test.pl
#-----
use My::Utils;
print "Content-type: text/plain\n\n";
my $sub_int = \&colour;
my $sub_ext = \&My::Utils::colour;
print "int $sub_int\n";
print "ext $sub_ext\n";
```

Issue a request, you will see something similar to:

```
int CODE(0x8510af8)
ext CODE(0x8510af8)
```

As you can see both point to the same CODE reference (meaning that it's the same symbol). After modifying *My/Utils.pm* again:

```
- sub colour { "red" }
+ sub colour { "blue" }
```

and calling the script on the second time, we get:

```
int CODE(0x8510af8)
ext CODE(0x851112c)
```

You can see that the internal CODE reference is not the same as the external one.

51.6.2.3 The Solution

There are two solutions to this problem:

Solution 1: replace `use ()` with an explicit `require() + import()`.

```
- use My::Utils;
+ require My::Utils; My::Utils->import();
```

now the changed functions will be reimported on every request.

Solution 2: remember to touch the script itself every time you change the module that it requires.

51.7 Threaded MPM and Multiple Perl Interpreters

If you use `Apache::Reload` with a threaded MPM and multiple Perl interpreters, the modules will be reloaded by each interpreter as they are used, not every interpreters at once. Similar to `mod_perl 1.0` where each child has its own Perl interpreter, the modules are reloaded as each child is hit with a request.

If a module is loaded at startup, the syntax tree of each subroutine is shared between interpreters (big win), but each subroutine has its own padlist (where lexical my variables are stored). Once `Apache::Reload` reloads a module, this sharing goes away and each Perl interpreter will have its own copy of the syntax tree for the reloaded subroutines.

51.8 Pseudo-hashes

The short summary of this is: Don't use pseudo-hashes. They are deprecated since Perl 5.8 and are removed in 5.9.

Use an array with constant indexes. Its faster in the general case, its more guaranteed, and generally, it works.

The long summary is that some work has been done to get this module working with modules that use pseudo-hashes, but it's still broken in the case of a single module that contains multiple packages that all use pseudo-hashes.

So don't do that.

51.9 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

51.10 Authors

Matt Sergeant, matt@sergeant.org

Stas Bekman (porting to mod_perl 2.0)

A few concepts borrowed from `Stonehenge::Reload` by Randal Schwartz and `Apache::StatINC` (mod_perl 1.x) by Doug MacEachern and Ask Bjoern Hansen.

51.11 See Also

`Stonehenge::Reload`

52 Apache::Status - Embedded interpreter status information

52.1 Synopsis

```
<Location /perl-status>
    SetHandler modperl
    PerlResponseHandler Apache::Status
</Location>
```

52.2 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>
    SetHandler modperl
    PerlResponseHandler Apache::Status
</Location>
```

Notice that under the "modperl" core handler the *Environment* menu option will show only the environment under that handler. To see the environment seen by handlers running under the "perl-script" core handler, configure `Apache::Status` as:

```
<Location /perl-status>
    SetHandler perl-script
    PerlResponseHandler 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 `PerlRequire` directives.

52.3 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

52.4 Prerequisites

The *Devel::Syndump* module, version **2.00** or higher.

52.5 Copyright

mod_perl 2.0 and its core modules are copyrighted under The Apache Software License, Version 1.1.

52.6 See Also

perl(1), Apache(3), Devel::Syndump(3), Data::Dumper(3), B(3), B::Graph(3), mod_perl 2.0 documentation.

52.7 Authors

Doug MacEachern with contributions from Stas Bekman

53 ModPerl::BuildMM -- A "subclass" of ModPerl::MM used for building mod_perl 2.0

53.1 SYNOPSIS

```
use ModPerl::BuildMM;

# ModPerl::BuildMM takes care of doing all the dirty job of overriding
ModPerl::BuildMM::WriteMakefile(...);

# if there is a need to extend the methods
sub MY::postamble {
    my $self = shift;

    my $string = $self->ModPerl::BuildMM::MY::postamble;

    $string .= "\nmydist : manifest tardist\n";

    return $string;
}
```

53.2 DESCRIPTION

ModPerl::BuildMM is a "subclass" of ModPerl::MM used for building mod_perl 2.0. Refer to ModPerl::MM manpage.

53.3 OVERRIDEN METHODS

ModPerl::BuildMM overrides the following methods:

53.3.1 ModPerl::BuildMM::MY::constants

53.3.2 ModPerl::BuildMM::MY::top_targets

53.3.3 ModPerl::BuildMM::MY::postamble

53.3.4 ModPerl::BuildMM::MY::post_initialize

53.3.5 ModPerl::BuildMM::MY::libscan

Table of Contents:

mod_perl APIs	1
Apache -- A ghost mod_perl 2.0 class	7
1 Apache -- A ghost mod_perl 2.0 class	7
1.1 Synopsis	8
1.2 Description	8
1.3 See Also	8
1.4 Copyright	8
1.5 Authors	8
Apache::Access - A Perl API for Apache request object	9
2 Apache::Access - A Perl API for Apache request object	9
2.1 Synopsis	10
2.2 Description	10
2.3 API	10
2.3.1 allow_options	10
2.3.2 allow_overrides	10
2.3.3 get_remote_logname	11
2.3.4 note_auth_failure	11
2.3.5 note_basic_auth_failure	11
2.3.6 note_digest_auth_failure	12
2.3.7 satisfies	12
2.3.8 some_auth_required	12
2.4 See Also	13
2.5 Copyright	13
2.6 Authors	13
Apache::CmdParms - Perl API for XXX	14
3 Apache::CmdParms - Perl API for XXX	14
3.1 Synopsis	15
3.2 Description	15
3.3 API	15
3.3.1 info	15
3.3.2 override	15
3.3.3 limited	15
3.3.4 limited_xmethods	16
3.3.5 xlimited	16
3.3.6 config_file	16
3.3.7 directive	16
3.3.8 pool	17
3.3.9 temp_pool	17
3.3.10 server	17
3.3.11 path	18
3.3.12 cmd	18
3.3.13 context	18
3.3.14 err_directive	18
3.4 See Also	19

Table of Contents:

3.5 Copyright	19
3.6 Authors	19
Apache::Command - Perl API for XXX	20
4 Apache::Command - Perl API for XXX	20
4.1 Synopsis	21
4.2 Description	21
4.3 API	21
4.3.1 check_cmd_context	21
4.3.2 soak_end_container	21
4.3.3 next	22
4.3.4 name	22
4.3.5 cmd_data	22
4.3.6 req_override	22
4.3.7 args_how	23
4.3.8 errmsg	23
4.4 See Also	23
4.5 Copyright	23
4.6 Authors	23
Apache::compat -- 1.0 backward compatibility functions deprecated in 2.0	24
5 Apache::compat -- 1.0 backward compatibility functions deprecated in 2.0	24
5.1 Synopsis	25
5.2 Description	25
5.3 Compatibility Functions Colliding with mod_perl 2.0 API	25
5.3.1 Available Overridable Functions	26
5.4 Use in CPAN Modules	26
5.5 API	27
5.6 See Also	27
5.7 Copyright	27
5.8 Authors	27
Apache::Connection - Perl API for Apache connection object	28
6 Apache::Connection - Perl API for Apache connection object	28
6.1 Synopsis	29
6.2 Description	29
6.3 API	29
6.3.1 aborted	29
6.3.2 base_server	29
6.3.3 bucket_alloc	29
6.3.4 conn_config	30
6.3.5 id	30
6.3.6 input_filters	30
6.3.7 keepalive	30
6.3.8 local_addr	30
6.3.9 local_host	31
6.3.10 local_ip	31
6.3.11 notes	31
6.3.12 output_filters	31
6.3.13 pool	32

6.3.14 remote_addr	32
6.3.15 remote_ip	32
6.3.16 remote_host	32
6.3.17 remote_logname	32
6.3.18 sbh	33
6.4 See Also	33
6.5 Copyright	33
6.6 Authors	33
Apache::Const - Perl Interface for Apache Constants	34
7 Apache::Const - Perl Interface for Apache Constants	34
7.1 Synopsis	35
7.2 Constants	35
7.2.1 :cmd_how	35
7.2.1.1 Apache::FLAG	35
7.2.1.2 Apache::ITERATE	35
7.2.1.3 Apache::ITERATE2	35
7.2.1.4 Apache::NO_ARGS	35
7.2.1.5 Apache::RAW_ARGS	35
7.2.1.6 Apache::TAKE1	35
7.2.1.7 Apache::TAKE12	35
7.2.1.8 Apache::TAKE123	35
7.2.1.9 Apache::TAKE13	35
7.2.1.10 Apache::TAKE2	35
7.2.1.11 Apache::TAKE23	35
7.2.1.12 Apache::TAKE3	35
7.2.2 :common	35
7.2.2.1 Apache::AUTH_REQUIRED	36
7.2.2.2 Apache::DECLINED	36
7.2.2.3 Apache::DONE	36
7.2.2.4 Apache::FORBIDDEN	36
7.2.2.5 Apache::NOT_FOUND	36
7.2.2.6 Apache::OK	36
7.2.2.7 Apache::REDIRECT	36
7.2.2.8 Apache::SERVER_ERROR	36
7.2.3 :config	36
7.2.3.1 Apache::DECLINE_CMD	36
7.2.4 :filter_type	36
7.2.4.1 Apache::FTYPE_CONNECTION	36
7.2.4.2 Apache::FTYPE_CONTENT_SET	36
7.2.4.3 Apache::FTYPE_NETWORK	36
7.2.4.4 Apache::FTYPE_PROTOCOL	36
7.2.4.5 Apache::FTYPE_RESOURCE	36
7.2.4.6 Apache::FTYPE_TRANSCODE	36
7.2.5 :http	37
7.2.5.1 Apache::HTTP_ACCEPTED	37
7.2.5.2 Apache::HTTP_BAD_GATEWAY	37
7.2.5.3 Apache::HTTP_BAD_REQUEST	37

Table of Contents:

7.2.5.4	Apache::HTTP_CONFLICT	37
7.2.5.5	Apache::HTTP_CONTINUE	37
7.2.5.6	Apache::HTTP_CREATED	37
7.2.5.7	Apache::HTTP_EXPECTATION_FAILED	37
7.2.5.8	Apache::HTTP_FAILED_DEPENDENCY	37
7.2.5.9	Apache::HTTP_FORBIDDEN	37
7.2.5.10	Apache::HTTP_GATEWAY_TIME_OUT	37
7.2.5.11	Apache::HTTP_GONE	37
7.2.5.12	Apache::HTTP_INSUFFICIENT_STORAGE	37
7.2.5.13	Apache::HTTP_INTERNAL_SERVER_ERROR	37
7.2.5.14	Apache::HTTP_LENGTH_REQUIRED	37
7.2.5.15	Apache::HTTP_LOCKED	37
7.2.5.16	Apache::HTTP_METHOD_NOT_ALLOWED	37
7.2.5.17	Apache::HTTP_MOVED_PERMANENTLY	37
7.2.5.18	Apache::HTTP_MOVED_TEMPORARILY	37
7.2.5.19	Apache::HTTP_MULTIPLE_CHOICES	37
7.2.5.20	Apache::HTTP_MULTI_STATUS	38
7.2.5.21	Apache::HTTP_NON_AUTHORITATIVE	38
7.2.5.22	Apache::HTTP_NOT_ACCEPTABLE	38
7.2.5.23	Apache::HTTP_NOT_EXTENDED	38
7.2.5.24	Apache::HTTP_NOT_FOUND	38
7.2.5.25	Apache::HTTP_NOT_IMPLEMENTED	38
7.2.5.26	Apache::HTTP_NOT_MODIFIED	38
7.2.5.27	Apache::HTTP_NO_CONTENT	38
7.2.5.28	Apache::HTTP_OK	38
7.2.5.29	Apache::HTTP_PARTIAL_CONTENT	38
7.2.5.30	Apache::HTTP_PAYMENT_REQUIRED	38
7.2.5.31	Apache::HTTP_PRECONDITION_FAILED	38
7.2.5.32	Apache::HTTP_PROCESSING	38
7.2.5.33	Apache::HTTP_PROXY_AUTHENTICATION_REQUIRED	38
7.2.5.34	Apache::HTTP_RANGE_NOT_SATISFIABLE	38
7.2.5.35	Apache::HTTP_REQUEST_ENTITY_TOO_LARGE	38
7.2.5.36	Apache::HTTP_REQUEST_TIME_OUT	38
7.2.5.37	Apache::HTTP_REQUEST_URI_TOO_LARGE	38
7.2.5.38	Apache::HTTP_RESET_CONTENT	38
7.2.5.39	Apache::HTTP_SEE_OTHER	38
7.2.5.40	Apache::HTTP_SERVICE_UNAVAILABLE	38
7.2.5.41	Apache::HTTP_SWITCHING_PROTOCOLS	39
7.2.5.42	Apache::HTTP_TEMPORARY_REDIRECT	39
7.2.5.43	Apache::HTTP_UNAUTHORIZED	39
7.2.5.44	Apache::HTTP_UNPROCESSABLE_ENTITY	39
7.2.5.45	Apache::HTTP_UNSUPPORTED_MEDIA_TYPE	39
7.2.5.46	Apache::HTTP_USE_PROXY	39
7.2.5.47	Apache::HTTP_VARIANT_ALSO_VARIES	39
7.2.6	:input_mode	39
7.2.6.1	Apache::MODE_EATCRLF	39
7.2.6.2	Apache::MODE_EXHAUSTIVE	39

7.2.6.3	Apache::MODE_GETLINE	39
7.2.6.4	Apache::MODE_INIT	39
7.2.6.5	Apache::MODE_READBYTES	39
7.2.6.6	Apache::MODE_SPECULATIVE	39
7.2.7	:log	39
7.2.7.1	Apache::LOG_ALERT	39
7.2.7.2	Apache::LOG_CRIT	39
7.2.7.3	Apache::LOG_DEBUG	39
7.2.7.4	Apache::LOG_EMERG	40
7.2.7.5	Apache::LOG_ERR	40
7.2.7.6	Apache::LOG_INFO	40
7.2.7.7	Apache::LOG_LEVELMASK	40
7.2.7.8	Apache::LOG_NOTICE	40
7.2.7.9	Apache::LOG_STARTUP	40
7.2.7.10	Apache::LOG_TOCLIENT	40
7.2.7.11	Apache::LOG_WARNING	40
7.2.8	:methods	40
7.2.8.1	Apache::METHODS	40
7.2.8.2	Apache::M_BASELINE_CONTROL	40
7.2.8.3	Apache::M_CHECKIN	40
7.2.8.4	Apache::M_CHECKOUT	40
7.2.8.5	Apache::M_CONNECT	40
7.2.8.6	Apache::M_COPY	40
7.2.8.7	Apache::M_DELETE	40
7.2.8.8	Apache::M_GET	40
7.2.8.9	Apache::M_INVALID	40
7.2.8.10	Apache::M_LABEL	40
7.2.8.11	Apache::M_LOCK	41
7.2.8.12	Apache::M_MERGE	41
7.2.8.13	Apache::M_MKACTIVITY	41
7.2.8.14	Apache::M_MKCOL	41
7.2.8.15	Apache::M_MKWORKSPACE	41
7.2.8.16	Apache::M_MOVE	41
7.2.8.17	Apache::M_OPTIONS	41
7.2.8.18	Apache::M_PATCH	41
7.2.8.19	Apache::M_POST	41
7.2.8.20	Apache::M_PROPFIND	41
7.2.8.21	Apache::M_PROPPATCH	41
7.2.8.22	Apache::M_PUT	41
7.2.8.23	Apache::M_REPORT	41
7.2.8.24	Apache::M_TRACE	41
7.2.8.25	Apache::M_UNCHECKOUT	41
7.2.8.26	Apache::M_UNLOCK	41
7.2.8.27	Apache::M_UPDATE	41
7.2.8.28	Apache::M_VERSION_CONTROL	41
7.2.9	:mpmq	41
7.2.9.1	Apache::MPMQ_NOT_SUPPORTED	42

Table of Contents:

7.2.9.2	Apache::MPMQ_STATIC	42
7.2.9.3	Apache::MPMQ_DYNAMIC	42
7.2.9.4	Apache::MPMQ_MAX_DAEMON_USED	42
7.2.9.5	Apache::MPMQ_IS_THREADED	42
7.2.9.6	Apache::MPMQ_IS_FORKED	42
7.2.9.7	Apache::MPMQ_HARD_LIMIT_DAEMONS	42
7.2.9.8	Apache::MPMQ_HARD_LIMIT_THREADS	42
7.2.9.9	Apache::MPMQ_MAX_THREADS	42
7.2.9.10	Apache::MPMQ_MIN_SPARE_DAEMONS	42
7.2.9.11	Apache::MPMQ_MIN_SPARE_THREADS	42
7.2.9.12	Apache::MPMQ_MAX_SPARE_DAEMONS	42
7.2.9.13	Apache::MPMQ_MAX_SPARE_THREADS	42
7.2.9.14	Apache::MPMQ_MAX_REQUESTS_DAEMON	42
7.2.9.15	Apache::MPMQ_MAX_DAEMONS	42
7.2.10	:options	42
7.2.10.1	Apache::OPT_ALL	42
7.2.10.2	Apache::OPT_EXECCGI	42
7.2.10.3	Apache::OPT_INCLUDES	42
7.2.10.4	Apache::OPT_INCNOEXEC	43
7.2.10.5	Apache::OPT_INDEXES	43
7.2.10.6	Apache::OPT_MULTI	43
7.2.10.7	Apache::OPT_NONE	43
7.2.10.8	Apache::OPT_SYM_LINKS	43
7.2.10.9	Apache::OPT_SYM_OWNER	43
7.2.10.10	Apache::OPT_UNSET	43
7.2.11	:override	43
7.2.11.1	Apache::ACCESS_CONF	43
7.2.11.2	Apache::OR_ALL	43
7.2.11.3	Apache::OR_AUTHCFG	43
7.2.11.4	Apache::OR_FILEINFO	43
7.2.11.5	Apache::OR_INDEXES	43
7.2.11.6	Apache::OR_LIMIT	43
7.2.11.7	Apache::OR_NONE	43
7.2.11.8	Apache::OR_OPTIONS	43
7.2.11.9	Apache::OR_UNSET	43
7.2.11.10	Apache::RSRC_CONF	43
7.2.12	:platform	43
7.2.12.1	Apache::CRLF	44
7.2.12.2	Apache::CR	44
7.2.12.3	Apache::LF	44
7.2.13	:remotehost	44
7.2.13.1	Apache::REMOTE_DOUBLE_REV	44
7.2.13.2	Apache::REMOTE_HOST	44
7.2.13.3	Apache::REMOTE_NAME	44
7.2.13.4	Apache::REMOTE_NOLOOKUP	44
7.2.14	:satisfy	44
7.2.14.1	Apache::SATISFY_ALL	44

7.2.14.2	Apache::SATISFY_ANY	44
7.2.14.3	Apache::SATISFY_NOSPEC	44
7.2.15	:types	44
7.2.15.1	Apache::DIR_MAGIC_TYPE	44
7.3	See Also	44
7.4	Copyright	45
7.5	Authors	45
	Apache::Directive - Perl API for manipulating Apache configuration tree	46
8	Apache::Directive - Perl API for manipulating Apache configuration tree	46
8.1	Synopsis	47
8.2	Description	47
8.3	Class Methods	47
8.3.1	conftree()	48
8.4	Object Methods	48
8.4.1	as_hash()	48
8.4.2	as_string()	48
8.4.3	lookup()	48
8.4.4	walk_config	49
8.4.5	directive	50
8.4.6	args	50
8.4.7	next	50
8.4.8	first_child	50
8.4.9	parent	51
8.4.10	data	51
8.4.11	filename	51
8.4.12	line_num	51
8.5	See Also	52
8.6	Copyright	52
8.7	Authors	52
	Apache::Filter - Perl API for Apache 2.0 Filtering	53
9	Apache::Filter - Perl API for Apache 2.0 Filtering	53
9.1	Synopsis	54
9.2	Description	54
9.3	Common Filter API	54
9.3.1	c	54
9.3.2	ctx	54
9.3.3	frec	55
9.3.4	next	55
9.3.5	r	56
9.3.6	remove	56
9.4	Bucket Brigade Filter API	56
9.4.1	fflush	57
9.4.2	get_brigade	57
9.4.3	pass_brigade	58
9.5	Streaming Filter API	59
9.5.1	seen_eos	59
9.5.2	read	60

Table of Contents:

9.5.3	fputs	60
9.5.4	print	60
9.6	Other Filter-related API	61
9.6.1	add_input_filter	61
9.6.2	add_output_filter	61
9.7	TIE Interface	61
9.7.1	TIEHANDLE	62
9.7.2	PRINT	62
9.8	Filter Handler Attributes	62
9.8.1	FilterRequestHandler	62
9.8.2	FilterConnectionHandler	63
9.8.3	FilterInitHandler	63
9.8.4	FilterHasInitHandler	63
9.9	Configuration	64
9.9.1	PerlInputFilterHandler	64
9.9.2	PerlOutputFilterHandler	64
9.9.3	PerlSetInputFilter	64
9.9.4	PerlSetOutputFilter	64
9.10	See Also	64
9.11	Copyright	64
9.12	Authors	64
Apache::FilterRec - Perl API for manipulating the Apache filter record		65
10	Apache::FilterRec - Perl API for manipulating the Apache filter record	65
10.1	Synopsis	66
10.2	Description	66
10.3	API	66
10.3.1	name	66
10.3.2	next	66
10.4	See Also	66
10.5	Copyright	66
10.6	Authors	67
Apache::HookRun - Perl API for XXX		68
11	Apache::HookRun - Perl API for XXX	68
11.1	Synopsis	69
11.2	Description	69
11.3	API	69
11.3.1	die	69
11.3.2	invoke_handler	69
11.3.3	run_access_checker	70
11.3.4	run_auth_checker	70
11.3.5	run_check_user_id	70
11.3.6	run_create_request	71
11.3.7	run_fixups	71
11.3.8	run_handler	71
11.3.9	run_header_parser	72
11.3.10	run_log_transaction	72
11.3.11	run_map_to_storage	73

11.3.12 run_post_read_request	73
11.3.13 run_translate_name	73
11.3.14 run_type_checker	74
11.4 See Also	74
11.5 Copyright	74
11.6 Authors	74
Apache::Log - Perl API for Apache Logging Methods	75
12 Apache::Log - Perl API for Apache Logging Methods	75
12.1 Synopsis	76
12.2 Description	77
12.3 Constants	77
12.3.1 LogLevel Constants	77
12.3.1.1 Apache::LOG_EMERG	77
12.3.1.2 Apache::LOG_ALERT	78
12.3.1.3 Apache::LOG_CRIT	78
12.3.1.4 Apache::LOG_ERR	78
12.3.1.5 Apache::LOG_WARNING	78
12.3.1.6 Apache::LOG_NOTICE	78
12.3.1.7 Apache::LOG_INFO	78
12.3.1.8 Apache::LOG_DEBUG	78
12.3.2 Other Constants	78
12.3.2.1 Apache::LOG_LEVELMASK	78
12.3.2.2 Apache::LOG_TOCLIENT	78
12.3.2.3 Apache::LOG_STARTUP	79
12.4 Server Logging Methods	79
12.4.1 \$s->log_error	79
12.4.2 \$s->log_serror	79
12.4.3 \$s->log	80
12.5 Request Logging Methods	80
12.5.1 \$r->log_error	80
12.5.2 \$r->log_rerror	81
12.5.3 \$r->log	81
12.6 Other Logging Methods	81
12.6.1 LogLevel Methods	81
12.6.2 emerg	82
12.6.3 alert	82
12.6.4 crit	82
12.6.5 error	82
12.6.6 warn	82
12.6.7 notice	82
12.6.8 info	82
12.6.9 debug	83
12.7 General Functions	83
12.7.1 Apache::Log::LOG_MARK	83
12.7.2 Apache::Log::log_pid	83
12.8 Aliases	83
12.8.1 \$s->warn	83

Table of Contents:

12.8.2	Apache->warn	84
12.8.3	Apache::warn	84
12.9	See Also	84
12.10	Copyright	84
12.11	Authors	84
	Apache::Module - Perl API for creating and working with Apache modules	85
13	Apache::Module - Perl API for creating and working with Apache modules	85
13.1	Synopsis	86
13.2	Description	86
13.3	API	86
13.3.1	find_linked_module	86
13.3.2	find_module_name	86
13.3.3	remove_loaded_module	87
13.3.4	remove_module	87
13.3.5	top_module	87
13.3.6	version	87
13.3.7	minor_version	88
13.3.8	module_index	88
13.3.9	name	88
13.3.10	dynamic_load_handle	88
13.3.11	next	89
13.3.12	cmds	89
13.4	See Also	89
13.5	Copyright	89
13.6	Authors	89
	Apache::PerlSections - Default Handler for Perl sections	90
14	Apache::PerlSections - Default Handler for Perl sections	90
14.1	Synopsis	91
14.2	Description	91
14.3	Configuration Variables	92
14.3.1	\$Apache::Server::SaveConfig	92
14.3.2	\$Apache::Server::StrictPerlSections	93
14.4	Advanced API	93
14.5	Bugs	94
14.5.1	<Perl> directive missing closing '>'	94
14.6	See Also	94
14.7	Copyright	94
14.8	Authors	94
	Apache::Process - Perl API for XXX	95
15	Apache::Process - Perl API for XXX	95
15.1	Synopsis	96
15.2	Description	96
15.3	API	96
15.3.1	pool	96
15.3.2	pconf	96
15.3.3	short_name	96
15.4	See Also	97

15.5 Copyright	97
15.6 Authors	97
Apache::RequestIO - Perl API for Apache request record IO	98
16 Apache::RequestIO - Perl API for Apache request record IO	98
16.1 Synopsis	99
16.2 Description	99
16.3 API	99
16.3.1 discard_request_body	99
16.3.2 setup_client_block	99
16.3.3 should_client_block	100
16.3.4 print	100
16.3.5 read	101
16.3.6 rflush	101
16.3.7 sendfile	101
16.3.8 write	101
16.4 TIE Interface	102
16.4.1 OPEN	102
16.4.2 UNTIE	102
16.4.3 PRINTF	102
16.4.4 CLOSE	102
16.4.5 PRINT	103
16.4.6 BINMODE	103
16.4.7 WRITE	103
16.4.8 TIEHANDLE	103
16.4.9 READ	104
16.5 See Also	104
16.6 Copyright	104
16.7 Authors	104
Apache::RequestRec - Perl API for Apache request record accessors	105
17 Apache::RequestRec - Perl API for Apache request record accessors	105
17.1 Synopsis	106
17.2 Description	106
17.3 API	106
17.3.1 proxyreq	106
17.3.2 pool	106
17.3.3 connection	107
17.3.4 server	107
17.3.5 next	107
17.3.6 prev	107
17.3.7 main	107
17.3.8 the_request	108
17.3.9 assbackwards	108
17.3.10 header_only	108
17.3.11 protocol	109
17.3.12 proto_num	109
17.3.13 hostname	109
17.3.14 request_time	109

Table of Contents:

17.3.15	status_line	110
17.3.16	status	110
17.3.17	method	110
17.3.18	method_number	110
17.3.19	allowed	111
17.3.20	allowed_xmethods	111
17.3.21	allowed_methods	111
17.3.22	bytes_sent	112
17.3.23	mtime	112
17.3.24	remaining	112
17.3.25	headers_in	113
17.3.26	headers_out	113
17.3.27	err_headers_out	113
17.3.28	notes	113
17.3.29	handler	114
17.3.30	content_encoding	114
17.3.31	content_languages	114
17.3.32	user	114
17.3.33	ap_auth_type	115
17.3.34	no_local_copy	115
17.3.35	unparsed_uri	115
17.3.36	uri	116
17.3.37	filename	116
17.3.38	canonical_filename	116
17.3.39	path_info	116
17.3.40	args	117
17.3.41	used_path_info	117
17.3.42	per_dir_config	117
17.3.43	request_config	117
17.3.44	output_filters	118
17.3.45	input_filters	118
17.3.46	proto_output_filters	118
17.3.47	proto_input_filters	118
17.4	See Also	119
17.5	Copyright	119
17.6	Authors	119
Apache::RequestUtil - Perl API for Apache request record utils		120
18	Apache::RequestUtil - Perl API for Apache request record utils	120
18.1	Synopsis	121
18.2	Description	121
18.3	Functions API	121
18.3.1	Apache->request()	121
18.4	Methods API	121
18.4.1	default_type	121
18.4.2	document_root	121
18.4.3	get_limit_req_body	122
18.4.4	get_server_name	122

18.4.5	get_server_port	122
18.4.6	is_initial_req	123
18.4.7	add_config	123
18.4.8	location	123
18.4.9	location_merge	123
18.4.10	pnotes	124
18.4.11	no_cache	124
18.4.12	as_string	124
18.4.13	get_handlers	124
18.4.14	push_handlers	125
18.4.15	set_handlers	125
18.4.16	set_basic_credentials	126
18.4.17	slurp_filename	126
18.4.18	is_perl_option_enabled	127
18.4.19	dir_config	127
18.5	See Also	128
18.6	Copyright	128
18.7	Authors	128
Apache::Response - Perl API for Apache HTTP request response methods		129
19	Apache::Response - Perl API for Apache HTTP request response methods	129
19.1	Synopsis	130
19.2	Description	130
19.3	API	130
19.3.1	custom_response	130
19.3.2	make_etag	130
19.3.3	meets_conditions	131
19.3.4	rationalize_mtime	131
19.3.5	send_error_response	132
19.3.6	send_mmap	132
19.3.7	set_content_length	133
19.3.8	set_etag	133
19.3.9	set_keepalive	133
19.3.10	update_mtime	134
19.3.11	set_last_modified	134
19.3.12	send_cgi_header	134
19.4	See Also	134
19.5	Copyright	134
19.6	Authors	135
Apache::Server - Perl API for for Apache server record accessors		136
20	Apache::Server - Perl API for for Apache server record accessors	136
20.1	Synopsis	137
20.2	Description	137
20.3	API	137
20.3.1	process	137
20.3.2	next	137
20.3.3	server_admin	138
20.3.4	server_hostname	138

Table of Contents:

20.3.5	port	138
20.3.6	error_fname	139
20.3.7	loglevel	139
20.3.8	is_virtual	140
20.3.9	module_config	140
20.3.10	lookup_defaults	140
20.3.11	addrs	141
20.3.12	timeout	141
20.3.13	keep_alive_timeout	142
20.3.14	keep_alive_max	142
20.3.15	keep_alive	143
20.3.16	path	143
20.3.17	names	143
20.3.18	wild_names	144
20.3.19	limit_req_line	144
20.3.20	limit_req_fieldsize	145
20.3.21	limit_req_fields	145
20.4	See Also	145
20.5	Copyright	146
20.6	Authors	146
Apache::ServerUtil - Perl API for XXX		147
21	Apache::ServerUtil - Perl API for XXX	147
21.1	Synopsis	148
21.2	Description	148
21.3	Constants	148
21.3.1	Apache::Server::server_root	148
21.4	Functions API	148
21.4.1	add_version_component	148
21.4.2	exists_config_define	149
21.4.3	get_server_built	149
21.4.4	get_server_version	149
21.5	Methods API	149
21.5.1	server_root_relative()	150
21.5.2	error_log2stderr	150
21.5.3	psignature	150
21.5.4	dir_config	151
21.5.5	is_perl_option_enabled	152
21.5.6	get_handlers	152
21.5.7	push_handlers	152
21.5.8	set_handlers	153
21.5.9	method_register	154
21.5.10	get_status_line	154
21.5.11	server	154
21.6	See Also	155
21.7	Copyright	155
21.8	Authors	155

Apache::SubProcess -- Executing SubProcesses from mod_perl	156
22 Apache::SubProcess -- Executing SubProcesses from mod_perl	156
22.1 Synopsis	157
22.2 Description	157
22.3 API	157
22.3.1 spawn_proc_prog	157
22.4 See Also	158
22.5 Copyright	158
22.6 Authors	158
Apache::SubRequest - Perl API for Apache subrequests	159
23 Apache::SubRequest - Perl API for Apache subrequests	159
23.1 Synopsis	160
23.2 Description	160
23.3 API	160
23.3.1 DESTROY	160
23.3.2 internal_fast_redirect	160
23.3.3 internal_redirect	161
23.3.4 internal_redirect_handler	161
23.3.5 lookup_dirent	161
23.3.6 lookup_file	162
23.3.7 lookup_uri	163
23.3.8 lookup_method_uri	163
23.4 See Also	164
23.5 Copyright	164
23.6 Authors	164
Apache::URI - Perl API for manipulating URIs	165
24 Apache::URI - Perl API for manipulating URIs	165
24.1 Synopsis	166
24.2 Description	166
24.3 API	166
24.3.1 construct_server	166
24.3.2 construct_url	166
24.3.3 parse_uri	167
24.4 See Also	167
24.5 Copyright	167
24.6 Authors	168
Apache::Util - Perl API for XXX	169
25 Apache::Util - Perl API for XXX	169
25.1 Synopsis	170
25.2 Description	170
25.3 Functions API	170
25.3.1 format_time	170
25.3.2 escape_path	170
25.4 See Also	171
25.5 Copyright	171
25.6 Authors	171

APR - Perl Interface for libapr and libaprutil Libraries	172
26 APR - Perl Interface for libapr and libaprutil Libraries	172
26.1 Synopsis	173
26.2 Description	173
26.3 See Also	173
26.4 Copyright	173
26.5 Authors	173
APR::Base64 - Perl API for XXX	174
27 APR::Base64 - Perl API for XXX	174
27.1 Synopsis	175
27.2 Description	175
27.3 API	175
27.3.1 encode_len	175
27.4 See Also	175
27.5 Copyright	175
27.6 Authors	175
APR::Brigade - Perl API for XXX	176
28 APR::Brigade - Perl API for XXX	176
28.1 Synopsis	177
28.2 Description	177
28.3 API	177
28.3.1 destroy	177
28.3.2 split	177
28.3.3 concat	178
28.3.4 empty	178
28.3.5 insert_head	178
28.3.6 insert_tail	178
28.4 See Also	179
28.5 Copyright	179
28.6 Authors	179
APR::Bucket - Perl API for XXX	180
29 APR::Bucket - Perl API for XXX	180
29.1 Synopsis	181
29.2 Description	181
29.3 API	181
29.3.1 eos_create	181
29.3.2 flush_create	181
29.3.3 insert_after	182
29.3.4 insert_before	182
29.3.5 is_eos	182
29.3.6 is_flush	182
29.3.7 remove	183
29.4 See Also	183
29.5 Copyright	183
29.6 Authors	183

APR::Const - Perl Interface for APR Constants	184
30 APR::Const - Perl Interface for APR Constants	184
30.1 Synopsis	185
30.2 Constants	185
30.2.1 :common	185
30.2.1.1 APR::SUCCESS	185
30.2.2 :error	185
30.2.2.1 APR::EABOVEROOT	185
30.2.2.2 APR::EABSOLUTE	185
30.2.2.3 APR::EACCES	185
30.2.2.4 APR::EAGAIN	185
30.2.2.5 APR::EBADDATE	185
30.2.2.6 APR::EBADF	185
30.2.2.7 APR::EBADIP	185
30.2.2.8 APR::EBADMASK	185
30.2.2.9 APR::EBADPATH	185
30.2.2.10 APR::EBUSY	185
30.2.2.11 APR::ECONNABORTED	185
30.2.2.12 APR::ECONNREFUSED	186
30.2.2.13 APR::ECONNRESET	186
30.2.2.14 APR::EDSOOPEN	186
30.2.2.15 APR::EEXIST	186
30.2.2.16 APR::EFTYPE	186
30.2.2.17 APR::EGENERAL	186
30.2.2.18 APR::EHOSTUNREACH	186
30.2.2.19 APR::EINCOMPLETE	186
30.2.2.20 APR::EINIT	186
30.2.2.21 APR::EINPROGRESS	186
30.2.2.22 APR::EINTR	186
30.2.2.23 APR::EINVAL	186
30.2.2.24 APR::EINVALSOCK	186
30.2.2.25 APR::EMFILE	186
30.2.2.26 APR::EMISMATCH	186
30.2.2.27 APR::ENAMETOOLONG	186
30.2.2.28 APR::END	186
30.2.2.29 APR::ENETUNREACH	186
30.2.2.30 APR::ENFILE	186
30.2.2.31 APR::ENODIR	186
30.2.2.32 APR::ENOENT	186
30.2.2.33 APR::ENOLOCK	187
30.2.2.34 APR::ENOMEM	187
30.2.2.35 APR::ENOPOLL	187
30.2.2.36 APR::ENOPOOL	187
30.2.2.37 APR::ENOPROC	187
30.2.2.38 APR::ENOSHMAVAIL	187
30.2.2.39 APR::ENOSOCKET	187
30.2.2.40 APR::ENOSPC	187

Table of Contents:

30.2.2.41	APR::ENOSTAT	187
30.2.2.42	APR::ENOTDIR	187
30.2.2.43	APR::ENOTEMPTY	187
30.2.2.44	APR::ENOTHDKEY	187
30.2.2.45	APR::ENOTHREAD	187
30.2.2.46	APR::ENOTIME	187
30.2.2.47	APR::ENOTIMPL	187
30.2.2.48	APR::ENOTSOCK	187
30.2.2.49	APR::EOF	187
30.2.2.50	APR::EPIPE	187
30.2.2.51	APR::ERELATIVE	187
30.2.2.52	APR::ESPIPE	187
30.2.2.53	APR::ETIMEDOUT	187
30.2.2.54	APR::EXDEV	188
30.2.3	:filemode	188
30.2.3.1	APR::BINARY	188
30.2.3.2	APR::BUFFERED	188
30.2.3.3	APR::CREATE	188
30.2.3.4	APR::DELONCLOSE	188
30.2.3.5	APR::EXCL	188
30.2.3.6	APR::PEND	188
30.2.3.7	APR::READ	188
30.2.3.8	APR::TRUNCATE	188
30.2.3.9	APR::WRITE	188
30.2.4	:filepath	188
30.2.4.1	APR::FILEPATH_NATIVE	188
30.2.4.2	APR::FILEPATH_NOTABOVEROOT	188
30.2.4.3	APR::FILEPATH_NOTABSOLUTE	188
30.2.4.4	APR::FILEPATH_NOTRELATIVE	188
30.2.4.5	APR::FILEPATH_SECUREROOT	188
30.2.4.6	APR::FILEPATH_SECUREROOTTEST	188
30.2.4.7	APR::FILEPATH_TRUENAME	189
30.2.5	:fileperms	189
30.2.5.1	APR::GEXECUTE	189
30.2.5.2	APR::GREAD	189
30.2.5.3	APR::GWRITE	189
30.2.5.4	APR::UEXECUTE	189
30.2.5.5	APR::UREAD	189
30.2.5.6	APR::UWRITE	189
30.2.5.7	APR::WEXECUTE	189
30.2.5.8	APR::WREAD	189
30.2.5.9	APR::WWRITE	189
30.2.6	:filetype	189
30.2.6.1	APR::NOFILE	189
30.2.6.2	APR::REG	189
30.2.6.3	APR::DIR	189
30.2.6.4	APR::CHR	189

30.2.6.5	APR::BLK	189
30.2.6.6	APR::PIPE	189
30.2.6.7	APR::LNK	190
30.2.6.8	APR::SOCK	190
30.2.6.9	APR::UNKFILE	190
30.2.7	:finfo	190
30.2.7.1	APR::FINFO_ETIME	190
30.2.7.2	APR::FINFO_CSIZE	190
30.2.7.3	APR::FINFO_CTIME	190
30.2.7.4	APR::FINFO_DEV	190
30.2.7.5	APR::FINFO_DIRENT	190
30.2.7.6	APR::FINFO_GPROT	190
30.2.7.7	APR::FINFO_GROUP	190
30.2.7.8	APR::FINFO_ICASE	190
30.2.7.9	APR::FINFO_IDENT	190
30.2.7.10	APR::FINFO_INODE	190
30.2.7.11	APR::FINFO_LINK	190
30.2.7.12	APR::FINFO_MIN	190
30.2.7.13	APR::FINFO_MTIME	190
30.2.7.14	APR::FINFO_NAME	190
30.2.7.15	APR::FINFO_NLINK	190
30.2.7.16	APR::FINFO_NORM	191
30.2.7.17	APR::FINFO_OWNER	191
30.2.7.18	APR::FINFO_PROT	191
30.2.7.19	APR::FINFO_SIZE	191
30.2.7.20	APR::FINFO_TYPE	191
30.2.7.21	APR::FINFO_UPROT	191
30.2.7.22	APR::FINFO_USER	191
30.2.7.23	APR::FINFO_WPROT	191
30.2.8	:flock	191
30.2.8.1	APR::FLOCK_EXCLUSIVE	191
30.2.8.2	APR::FLOCK_NONBLOCK	191
30.2.8.3	APR::FLOCK_SHARED	191
30.2.8.4	APR::FLOCK_TYPEMASK	191
30.2.9	:hook	191
30.2.9.1	APR::HOOK_FIRST	191
30.2.9.2	APR::HOOK_LAST	191
30.2.9.3	APR::HOOK_MIDDLE	191
30.2.9.4	APR::HOOK_REALLY_FIRST	191
30.2.9.5	APR::HOOK_REALLY_LAST	192
30.2.10	:limit	192
30.2.10.1	APR::LIMIT_CPU	192
30.2.10.2	APR::LIMIT_MEM	192
30.2.10.3	APR::LIMIT_NOFILE	192
30.2.10.4	APR::LIMIT_NPROC	192
30.2.11	:lockmech	192
30.2.11.1	APR::LOCK_DEFAULT	192

Table of Contents:

30.2.11.2	APR::LOCK_FCNTL	192
30.2.11.3	APR::LOCK_FLOCK	192
30.2.11.4	APR::LOCK_POSIXSEM	192
30.2.11.5	APR::LOCK_PROC_PTHREAD	192
30.2.11.6	APR::LOCK_SYSVSEM	192
30.2.12	:poll	192
30.2.12.1	APR::POLLERR	192
30.2.12.2	APR::POLLHUP	192
30.2.12.3	APR::POLLIN	193
30.2.12.4	APR::POLLNVAL	193
30.2.12.5	APR::POLLOUT	193
30.2.12.6	APR::POLLPRI	193
30.2.13	:read_type	193
30.2.13.1	APR::BLOCK_READ	193
30.2.13.2	APR::NONBLOCK_READ	193
30.2.14	:shutdown_how	193
30.2.14.1	APR::SHUTDOWN_READ	193
30.2.14.2	APR::SHUTDOWN_READWRITE	193
30.2.14.3	APR::SHUTDOWN_WRITE	193
30.2.15	:socket	193
30.2.15.1	APR::SO_DEBUG	193
30.2.15.2	APR::SO_DISCONNECTED	193
30.2.15.3	APR::SO_KEEPAIVE	193
30.2.15.4	APR::SO_LINGER	193
30.2.15.5	APR::SO_NONBLOCK	194
30.2.15.6	APR::SO_RCVBUF	194
30.2.15.7	APR::SO_REUSEADDR	194
30.2.15.8	APR::SO_SNDBUF	194
30.2.16	:table	194
30.2.16.1	APR::OVERLAP_TABLES_MERGE	194
30.2.16.2	APR::OVERLAP_TABLES_SET	194
30.2.17	:uri	194
30.2.17.1	APR::URI_ACAP_DEFAULT_PORT	194
30.2.17.2	APR::URI_FTP_DEFAULT_PORT	194
30.2.17.3	APR::URI_GOPHER_DEFAULT_PORT	194
30.2.17.4	APR::URI_HTTPS_DEFAULT_PORT	194
30.2.17.5	APR::URI_HTTP_DEFAULT_PORT	194
30.2.17.6	APR::URI_IMAP_DEFAULT_PORT	194
30.2.17.7	APR::URI_LDAP_DEFAULT_PORT	194
30.2.17.8	APR::URI_NFS_DEFAULT_PORT	194
30.2.17.9	APR::URI_NNTP_DEFAULT_PORT	194
30.2.17.10	APR::URI_POP_DEFAULT_PORT	194
30.2.17.11	APR::URI_PROSPERO_DEFAULT_PORT	195
30.2.17.12	APR::URI_RTSP_DEFAULT_PORT	195
30.2.17.13	APR::URI_SIP_DEFAULT_PORT	195
30.2.17.14	APR::URI_SNEWS_DEFAULT_PORT	195
30.2.17.15	APR::URI_SSH_DEFAULT_PORT	195

30.2.17.16	APR::URI_TELNET_DEFAULT_PORT	195
30.2.17.17	APR::URI_TIP_DEFAULT_PORT	195
30.2.17.18	APR::URI_UNP_OMITPASSWORD	195
30.2.17.19	APR::URI_UNP_OMITPATHINFO	195
30.2.17.20	APR::URI_UNP_OMITQUERY	195
30.2.17.21	APR::URI_UNP_OMITSITEPART	195
30.2.17.22	APR::URI_UNP_OMITUSER	195
30.2.17.23	APR::URI_UNP_OMITUSERINFO	195
30.2.17.24	APR::URI_UNP_REVEALPASSWORD	195
30.2.17.25	APR::URI_WAIS_DEFAULT_PORT	195
30.3	See Also	195
30.4	Copyright	195
30.5	Authors	195
APR::Date - Perl API for XXX		196
31	APR::Date - Perl API for XXX	196
31.1	Synopsis	197
31.2	Description	197
31.3	API	197
31.3.1	parse_http	197
31.3.2	parse_rfc	197
31.4	See Also	198
31.5	Copyright	198
31.6	Authors	198
APR::Finfo - Perl API for XXX		199
32	APR::Finfo - Perl API for XXX	199
32.1	Synopsis	200
32.2	Description	200
32.3	API	200
32.3.1	stat	200
32.3.2	pool	200
32.3.3	valid	201
32.3.4	protection	201
32.3.5	filetype	201
32.3.6	user	201
32.3.7	group	202
32.3.8	inode	202
32.3.9	device	202
32.3.10	nlink	202
32.3.11	size	203
32.3.12	csize	203
32.3.13	atime	203
32.3.14	mtime	203
32.3.15	ctime	204
32.3.16	fname	204
32.3.17	name	204
32.4	See Also	204
32.5	Copyright	204

Table of Contents:

32.6 Authors	205
APR::NetLib - Perl API for XXX	206
33 APR::NetLib - Perl API for XXX	206
33.1 Synopsis	207
33.2 Description	207
33.3 API	207
33.3.1 test	207
33.4 See Also	207
33.5 Copyright	207
33.6 Authors	208
APR::PerlIO -- An APR Perl IO layer	209
34 APR::PerlIO -- An APR Perl IO layer	209
34.1 Synopsis	210
34.2 Description	210
34.3 Constants	210
34.3.1 APR::PerlIO::PERLIO_LAYERS_ENABLED	210
34.4 API	211
34.4.1 open	211
34.4.2 seek()	211
34.5 C API	212
34.6 See Also	212
34.7 Copyright	212
34.8 Authors	212
APR::Pool - Perl API for XXX	213
35 APR::Pool - Perl API for XXX	213
35.1 Synopsis	214
35.2 Description	214
35.3 API	214
35.3.1 cleanup_for_exec	214
35.3.2 clear	214
35.3.3 destroy	214
35.3.4 is_ancestor	215
35.3.5 tag	215
35.4 See Also	216
35.5 Copyright	216
35.6 Authors	216
APR::SockAddr - Perl API for XXX	217
36 APR::SockAddr - Perl API for XXX	217
36.1 Synopsis	218
36.2 Description	218
36.3 API	218
36.3.1 equal	218
36.4 See Also	218
36.5 Copyright	218
36.6 Authors	219

APR::Socket - Perl API for XXX	220
37 APR::Socket - Perl API for XXX	220
37.1 Synopsis	221
37.2 Description	221
37.3 API	221
37.3.1 bind	221
37.3.2 close	221
37.3.3 connect	222
37.3.4 listen	222
37.3.5 opt_get	222
37.3.6 opt_set	223
37.3.7 recvfrom	224
37.3.8 sendto	224
37.3.9 timeout_set	225
37.4 See Also	225
37.5 Copyright	225
37.6 Authors	226
APR::Table - Perl API for for manipulating opaque string-content table	227
38 APR::Table - Perl API for for manipulating opaque string-content table	227
38.1 Synopsis	228
38.2 Description	228
38.3 API	229
38.3.1 add	229
38.3.2 clear	229
38.3.3 compress	229
38.3.4 copy	230
38.3.5 do	230
38.3.6 get	231
38.3.7 make	231
38.3.8 merge	232
38.3.9 overlap	232
38.3.10 overlay	233
38.3.11 set	234
38.3.12 unset	234
38.4 TIE Interface	234
38.4.1 EXISTS	235
38.4.2 CLEAR	235
38.4.3 STORE	235
38.4.4 DELETE	235
38.4.5 FETCH	235
38.5 See Also	236
38.6 Copyright	236
38.7 Authors	236
APR::ThreadMutex - Perl API for XXX	237
39 APR::ThreadMutex - Perl API for XXX	237
39.1 Synopsis	238
39.2 Description	238

Table of Contents:

39.3 API	238
39.3.1 DESTROY	238
39.3.2 lock	238
39.3.3 pool_get	239
39.3.4 trylock	239
39.3.5 unlock	239
39.4 See Also	239
39.5 Copyright	240
39.6 Authors	240
APR::URI - Perl API for XXX	241
40 APR::URI - Perl API for XXX	241
40.1 Synopsis	242
40.2 Description	242
40.3 API	242
40.3.1 port_of_scheme	242
40.3.2 scheme	242
40.3.3 hostinfo	243
40.3.4 user	243
40.3.5 password	243
40.3.6 hostname	243
40.3.7 path	244
40.3.8 query	244
40.3.9 fragment	244
40.3.10 is_initialized	244
40.3.11 dns_looked_up	245
40.3.12 dns_resolved	245
40.4 See Also	245
40.5 Copyright	245
40.6 Authors	245
APR::Util - Perl API for XXX	246
41 APR::Util - Perl API for XXX	246
41.1 Synopsis	247
41.2 Description	247
41.3 API	247
41.3.1 filepath_name_get	247
41.3.2 password_get	247
41.4 See Also	248
41.5 Copyright	248
41.6 Authors	248
ModPerl::MethodLookup -- Map mod_perl 2.0 modules, objects and methods	249
42 ModPerl::MethodLookup -- Map mod_perl 2.0 modules, objects and methods	249
42.1 Synopsis	250
42.2 Description	250
42.3 API	251
42.3.1 lookup_method()	251
42.3.2 lookup_module()	252
42.3.3 lookup_object()	252

42.3.4	print_method()	253
42.3.5	print_module()	253
42.3.6	print_object()	253
42.3.7	preload_all_modules()	254
42.4	Applications	254
42.4.1	AUTOLOAD	254
42.4.2	Command Line Lookups	255
42.5	Todo	256
42.6	See Also	256
42.7	Author	256
ModPerl::MM -- A "subclass" of ExtUtils::MakeMaker for mod_perl 2.0		257
43	ModPerl::MM -- A "subclass" of ExtUtils::MakeMaker for mod_perl 2.0	257
43.1	Synopsis	258
43.2	Description	258
43.3	MY:: Default Methods	258
43.3.1	ModPerl::MM::MY::constants	259
43.3.2	ModPerl::MM::MY::post_initialize	259
43.4	WriteMakefile() Default Arguments	259
43.4.1	CCFLAGS	260
43.4.2	LIBS	260
43.4.3	INC	260
43.4.4	OPTIMIZE	260
43.4.5	LDDLFLAGS	260
43.4.6	TYPEMAPS	260
43.4.7	dynamic_lib	260
43.4.7.1	OTHERLDFLAGS	260
43.4.8	macro	260
43.4.8.1	MOD_INSTALL	260
43.5	Public API	260
43.5.1	WriteMakefile()	261
43.5.2	get_def_opt()	261
ModPerl::PerlRun - Run unaltered CGI scripts under mod_perl		262
44	ModPerl::PerlRun - Run unaltered CGI scripts under mod_perl	262
44.1	SYNOPSIS	263
44.2	DESCRIPTION	263
44.3	AUTHORS	263
44.4	SEE ALSO	263
ModPerl::Registry - Run unaltered CGI scripts persistently under mod_perl		264
45	ModPerl::Registry - Run unaltered CGI scripts persistently under mod_perl	264
45.1	Synopsis	265
45.2	Description	265
45.3	Security	266
45.4	Environment	266
45.5	Commandline Switches In First Line	266
45.6	Debugging	266
45.7	Caveats	266
45.8	Authors	267

45.9 See Also	267
ModPerl::RegistryBB - Run unaltered CGI scripts persistently under mod_perl	268
46 ModPerl::RegistryBB - Run unaltered CGI scripts persistently under mod_perl	268
46.1 Synopsis	269
46.2 Description	269
46.3 Authors	269
46.4 See Also	269
ModPerl::RegistryCooker - Cook mod_perl 2.0 Registry Modules	270
47 ModPerl::RegistryCooker - Cook mod_perl 2.0 Registry Modules	270
47.1 Synopsis	271
47.2 Description	271
47.2.1 Special Predefined Functions	273
47.3 Sub-classing Techniques	274
47.4 Examples	274
47.5 Authors	275
47.6 See Also	275
ModPerl::RegistryLoader - Compile ModPerl::RegistryCooker scripts at server startup	276
48 ModPerl::RegistryLoader - Compile ModPerl::RegistryCooker scripts at server startup	276
48.1 Synopsis	277
48.2 Description	277
48.3 Methods	277
48.4 Implementation Notes	280
48.5 Authors	280
48.6 SEE ALSO	280
ModPerl::Util - Helper mod_perl 2.0 Functions	281
49 ModPerl::Util - Helper mod_perl 2.0 Functions	281
49.1 Synopsis	282
49.2 Description	282
49.3 API	282
49.3.1 untaint	282
49.3.2 current_callback	282
49.3.3 exit	282
49.4 See Also	283
49.5 Copyright	283
49.6 Authors	283
Apache::porting -- a helper module for mod_perl 1.0 to mod_perl 2.0 porting	284
50 Apache::porting -- a helper module for mod_perl 1.0 to mod_perl 2.0 porting	284
50.1 Synopsis	285
50.2 Description	285
50.3 Culprits	285
50.4 See Also	286
50.5 Copyright	286
50.6 Authors	286
Apache::Reload - Reload Perl Modules when Changed on Disk	287
51 Apache::Reload - Reload Perl Modules when Changed on Disk	287
51.1 Synopsis	288
51.2 Description	288

51.2.1 Monitor All Modules in %INC	289
51.2.2 Register Modules Implicitly	289
51.2.3 Register Modules Explicitly	289
51.2.4 Monitor Only Certain Sub Directories	290
51.2.5 Special "Touch" File	290
51.3 Performance Issues	290
51.4 Debug	291
51.5 Silencing 'Constant subroutine ... redefined at' Warnings	291
51.6 Caveats	291
51.6.1 Problems With Reloading Modules Which Do Not Declare Their Package Name	291
51.6.2 Problems with Scripts Running with Registry Handlers that Cache the Code	292
51.6.2.1 The Problem	292
51.6.2.2 The Explanation	292
51.6.2.3 The Solution	293
51.7 Threaded MPM and Multiple Perl Interpreters	293
51.8 Pseudo-hashes	294
51.9 Copyright	294
51.10 Authors	294
51.11 See Also	294
Apache::Status - Embedded interpreter status information	295
52 Apache::Status - Embedded interpreter status information	295
52.1 Synopsis	296
52.2 Description	296
52.3 Options	296
52.4 Prerequisites	298
52.5 Copyright	298
52.6 See Also	298
52.7 Authors	299
ModPerl::BuildMM -- A "subclass" of ModPerl::MM used for building mod_perl 2.0	300
53 ModPerl::BuildMM -- A "subclass" of ModPerl::MM used for building mod_perl 2.0	300
53.1 SYNOPSIS	301
53.2 DESCRIPTION	301
53.3 OVERRIDEN METHODS	301
53.3.1 ModPerl::BuildMM::MY::constants	301
53.3.2 ModPerl::BuildMM::MY::top_targets	301
53.3.3 ModPerl::BuildMM::MY::postamble	301
53.3.4 ModPerl::BuildMM::MY::post_initialize	301
53.3.5 ModPerl::BuildMM::MY::libscan	301